THE UNIVERSITY *of* EDINBURGH

Instance compression of parametric problems and related hierarchies

Chiranjit Chakraborty

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2013

Abstract

We define instance compressibility ([13, 17]) for parametric problems in the classes PH and PSPACE. We observe that the problem Σ_iCIRCUITSAT of deciding satisfiability of a quantified Boolean circuit with $i-1$ alternations of quantifiers starting with an existential quantifier is complete for parametric problems in the class Σ_i^p with respect to w-reductions, and that analogously the problem QBCSAT (Quantified Boolean Circuit Satisfiability) is complete for parametric problems in PSPACE with respect to w-reductions. We show the following results about these problems:

1. If CIRCUITSAT is non-uniformly compressible within NP, then Σ_iCIRCUITSAT is non-uniformly compressible within NP, for any $i \geq 1$.

2. If QBCSAT is non-uniformly compressible (or even if satisfiability of quantified Boolean *CNF* formulae is non-uniformly compressible), then PSPACE $\subseteq$ NP/poly and PH collapses to the third level.

Next, we define Succinct Interactive Proof (Succinct IP) and by adapting the proof of IP = PSPACE ([11, 6]) , we show that QBCNFSAT (Quantified Boolean Formula (in *CNF*) Satisfiability) is in Succinct IP. On the contrary if QBCNFSAT has Succinct PCPs ([32]) , Polynomial Hierarchy (PH) collapses.

After extending the notion of instance compression to higher classes, we study the hierarchical structure of the parametric problems with respect to compressibility. For that purpose, we extend the existing definition of *VC*-hierarchy ([13]) to parametric problems. After that, we have considered a long list of natural NP problems and tried to classify them into some level of *VC*-hierarchy. We have shown some of the new w-reductions in this context and pointed out a few interesting results including the ones as follows.

1. CLIQUE is VC_1-complete (using the results in [14]).

2. SET SPLITTING and NAE-SAT are VC_2-complete.

We have also introduced a new complexity class VC_E in this context and showed some hardness and completeness results for this class. We have done a comparison of *VC*-hierarchy with other related hierarchies in parameterized complexity domain as well.

Next, we define the compression of counting problems and the analogous classification of them with respect to the notion of instance compression. We define #*VC*-hierarchy for this purpose and similarly classify a large number of natural counting problems with respect to this hierarchy, by showing some interesting hardness and completeness results.

We have considered some of the interesting practical problems as well other than popular NP problems (e.g., #MULTICOLOURED CLIQUE, #SELECTED DOMINATING SET etc.) and studied their complexity for both decision and counting version. We have also considered a large variety of circuit satisfiability problems (e.g., #MONOTONE WEIGHTED-CNFSAT, #EXACT DNF-SAT etc.) and proved some interesting results about them with respect to the theory of instance compressibility.

Acknowledgements

I would like to thank to my PhD advisor, Dr. Rahul Santhanam, for his continuous support to my PhD study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of my research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

I also like to thank my second supervisor, Dr. Kousha Etessami for his critical review and guidelines about my research. My sincere thanks also goes to Dr. Elham Kashefi for her very useful feedback in my yearly reviews.

Last but not the least, I would like to thank my parents, Anita Chakraborty and Dipak Chakraborty, for giving birth to me at the first place and supporting me throughout my life specially when I went through tough time.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Chiranjit Chakraborty*)

Table of Contents

Chapter 1

Introduction

1.1 Project aims and objectives

1.1.1 Introduction: Complexity Theory

Computational complexity theory is one of the popular branches of Theoretical Computer Science that deals with classifying computational problems with respect to their inherent difficulty ([48, 39]). In complexity theory we are particularly interested to solve the harder problems. Harder problems are classified in different classes. NP is one such complexity class we are mostly interested in. To deal with such difficult computational problems several well-known techniques are developed, such as, approximation algorithms, parametric complexity, sub-exponential algorithms, average-case complexity etc. Here we are specially interested to deal with such problems in a little bit different manner: *Compressibility*. Harnik and Naor [13] have introduced this concept mainly for the NP problems. Independently Bodlaender et al.[17] have developed a framework for analysing and studying polynomial kernelizability for natural parametric problems. Here we are not interested to find the direct solution of the problem instances, rather we are interested to know whether we can find a shorter instance with the same solution (*yes*/*no*) in an efficient manner. We emphasise that we are not interested to maintain the information about the problem instance, rather we are trying to maintain the solution. It is possible that we can find the solution of the problem in an efficient way and maintain the solution in just one bit (*yes*/*no*). Eventually, the output can be much shorter in size compared to the initial input if not as small as just one bit. Hence the term *compression* is used. So, initially we study the compressibility of NP problems. After that we are specifically interested to extend the idea to other

comparatively harder complexity classes.

In our work, any problem instance x has two important characteristics. One of them is instance size, (denoted by m) and another one is the length of the witness w or parameter (denoted by n). We are particularly interested in relatively harder problem instances for which instance size m is larger and witness length n relatively shorter but not too short. In particular, we are interested in those cases when $n << m$ and $m << 2^n$. When we will come to the formal definition and example (section 1.1.2) later it will be clearer why we are interested in these cases.

1.1.2 An example

To explain this particular setting in complexity theory, we are going to discuss it with an example. We start with SAT problem as it is one of the most popular NP problem. Let us consider the formal definition of the problem first.

SAT:

Input: A formula ϕ in *CNF* (conjunctive normal form) with n variables.

Task: Decide whether ϕ is satisfiable.

Parameter: n

As we can see in the definition, an instance ϕ for SAT problem is given to us. ϕ is a Boolean formula in conjunctive normal form over n variables. So, ϕ is satisfiable (i.e., *yes* SAT problem instance) if there exists an assignment to the n input variables such that every clause in ϕ has at least one *True* literal. In this example, suppose we are interested to compress the SAT instance (ϕ) to another shorter SAT instance. This kind of compression where we compress a problem instance to a shorter instance of same language, is termed as *proper-compression* or *self-compression*. In other cases, when after the compression the compressed instance is an instance of some other language L, is termed as *improper-compression*.

Here, our objective is to find a *self-compression algorithm* of SAT problem instances. So we ask, does there exist an efficient algorithm (running in time $poly(m)$) and a polynomial $p()$ with the following input and output?

Input: A formula ϕ in *CNF* with n variables where $|\phi| = m$.

Output: Another Boolean formula in *CNF* (say ψ) of size $poly(n)$ such that ϕ is satisfiable if and only if ψ is satisfiable.

The idea is, the length of the output formula ψ, should not be related to the length m of the original problem instance, but to the parameter of the problem which is number of variables n (which is the witness length for this problem). Any kind of positive or negative result related to this question of compression will be very interesting. We are going to discuss about it in more details soon.

It is already mentioned that we are interested about those problems for which n is much smaller than m. The reason behind this can be explained as follows. Our objective here is to take a problem instance of length m and produce another problem instance in polynomial time, whose size is polynomially bounded in witness length or parameter. At the same time, we also like to maintain the *yes*/*no* answer. If m and n are comparable (i.e., polynomially bounded to each other), the trivial identity function will be a compression function. Similarly, if n is too small, i.e., $m > 2^n$, we can search for every possible witnesses and find the solution in polynomial time. That is why we are particularly interested in non-trivial cases when $n << m$ and $m << 2^n$.

1.1.3 Primary objective behind this work

Harnik and Naor [13] introduced this concept mainly motivated by its cryptographic applications. They have also shown some applications related to complexity theory. Independently, Bodlaender et al. [17] also introduced similar concepts motivated by its application in the field of parameterized complexity. Our work is actually motivated by both of them. We have studied the importance of this notion in the classical complexity theory, as well as its implication in parameterized complexity domain. Besides, this approach has significance in some other fields as well. Using compression we can try to store many problem instances in much smaller space and so this approach can give us some ways to save space instead of storing many many big problem instances. But Fortnow and Santhanam showed ([32], Theorem 3.1) that the compressibility of the satisfiability problem for Boolean formulae (even non-uniformly) is unlikely, since it implies that the Polynomial Hierarchy (PH) collapses. Since then, there has been a very active stream of research extending this negative result to other problems in NP (e.g., [55]). Instance compressibility is a useful notion for structural complexity theory as well. We have discussed about that in the next section in more details.

1.2 Review of related works and primary motivation

As primary motivation behind this approach, Harnik and Naor [13] have mostly shown some of its cryptographic applications. But at the same time, this notion is closely related to the notion of polynomial kernelizability in parametrized complexity ([17, 47, 25]), which is motivated by algorithmic applications. However, in [13], the applications are shown in two opposite directions related to cryptography. They have shown that the constructions of some of the cryptographic primitives are possible using compression technique. But at the same time, they have pointed out the possibility of cryptanalysis using compression algorithms. It can be shown that for some applications in cryptography, the incompressibility of some of the languages is very much important. Some of the other related notions is also very much related to this idea of instance compression. Dubrov and Ishai [7] have shown the relevance of the notion of compression to derandomization. Dziembowski [53] shows that compression is related to the study of forward-secure storage. But our work is motivated mainly by its application in the field of complexity theory. Now we will see the specific application of the notion of compression in the field of complexity theory.

1.2.1 Part I: Implications to Parameterized Complexity Theory

The relationship between complexity and compression is always interesting and in general, has been studied as an interesting topic since the very early days of Complexity Theory. For example, concept of Kolmogorov Complexity [4] has similarities with the idea of instance compression. But, they have some fundamental differences. Here, we are not interested about the compressibility of the instances alone. We are interested to compress any instance maintaining the same solution, i.e., an *yes* or *no* instance will be mapped to an *yes* or *no* instance respectively (formal definition is in chapter 2). Hence, this work has some fundamental distinguishing features from several other related works. Primary aim in most of the previous works was how to retrieve the input of a compression algorithm, not the solution.

This idea of compression is rather mostly related to the idea of parameterized complexity (Downey and Fellows [44], Niedermeier [47], Flum and Grohe [25]). In parameterized complexity theory, we study the tractability of the problems when some parameter of the problem is fixed. This notion is known as *fixed parameter tractability* (FPT). *Kernelization* is known to be one of the basic techniques to find an efficient

algorithm in this context. We are now going to discuss about this in more details.

Parameterized complexity: In Parameterized complexity we ask about the complexity of NP problems based on some inherent parameter. For example, in satisfiability problem, the number of variables in a formula or in CLIQUE problem, clique size in a graph can be taken as parameter. In parameterized complexity, fixed-parameter tractability is an important concept as it specifies the property of tractable problems. Let us look at it in more formal way.

Definition 1. *A problem is fixed-parameter tractable (FPT) if it has an algorithm running in time $f(k)n^{O(1)}$ where n is the input size and $f(k)$ is an arbitrary function of the parameter k.*

Kernelization is one technique used to show fixed-parameter tractability of certain parametric problem. In this technique, one tries to find a reduction from the given instance to an instance whose size depends only on the parameter, maintaining the solution of the problem. This new reduced instance is known as *problem kernel*. Chen et al. [31] has shown that a problem is FPT if and only if it has a kernelization. But, this kernel is a function of the parameter and hence it can be arbitrarily large. So, we can basically interpret this kernelization in some other perspective to view the compression. Technically we can say that *polynomial kernelization* (where the *problem kernel* size is a polynomial function of the initial problem parameter) is equivalent to the deterministic compression to size $poly(n)$ (n is the parameter of the initial problem instance).

Because of the above mentioned similarities, the W-hierarchy [25] and other hierarchies in the field of parameterized complexity is very much connected to the VC-hierarchy [13]. This VC-hierarchy is a hierarchy which is defined depending on the notion of compression. (We have discussed about it in more details in chapter 3). Basically both are defined by the reductions to some kind of circuit problems. So, finding the solution to some of the open questions in compression, gives us some results in the field of Parameterized complexity as well. As our work very much related to the field of parameterized complexity, we will discuss about this in more details in the next section.

1.2.2 Part II: Implications to PCPs

Not only parameterized complexity, study of Probabilistically Checkable Proofs ([50, 49]) are also very much related to this notion of compressibility.

Probabilistically checkable proofs: Probabilistically checkable proofs or PCPs are one of the most popular field of studies in the field of complexity theory. The PCP theorem ([50, 49]) is known to be one of the famous landmark results in the field of complexity theory. It states that any decision problem in the complexity class NP has polynomial size proofs (polynomially bounded in problem instance size) that can be verified probabilistically by reading only a constant number of bits of the proof. Fortnow and Santhanam [32] have defined *succinct PCP*s, which are intuitively PCPs where the proof size depends polynomially on the witness length (parameter) rather than the length of the input. In the framework of parametric problems (formal definition is in chapter 2), this corresponds to the proof size to be dependent polynomially on the parameter rather than the length of the input. They have also shown that if SAT has succinct PCPs, then SAT is self-compressible with error less than 2^{-m}. So eventually it can be shown that SAT does not have succinct PCPs unless PH collapses. Besides, if SAT is self-compressible, then SAT has succinct PCPs [32]. All these results are very much important in complexity theory and keep us motivated to study further about this compression of hard problems.

1.2.3 Part III: Implication to Structural complexity and other related fields

Although Harnik and Naor said that finding compression algorithm will be a great help in the field of cryptography, some of the strong negative results related to this compression makes this field more popular in different domains of complexity theory. Fortnow and Santhanam [32] have shown that an error-less compression algorithm for SAT implies that the polynomial hierarchy will collapse. Very recently Andrew Drucker [2] has shown that compression of another interesting problem, AND-SAT is also impossible unless the polynomial hierarchy collapses. Not only that, he has shown some evidence against probabilistic compression (definition in [13, 32]) as well. Chen and Muller [58] pointed out that this generalises to compression with a one-sided error. These results restrict the application of error-free compression to construct collision

resistant hash functions as initially shown in [13]. But after such negative results, application of compressibility to different complexity theory branches becomes more prominent compared to cryptography. One such branch of complexity theory is Structural complexity.

Structural complexity: Structural complexity is a popular domain in complexity theory which studies the complexity classes directly, rather than the computational complexity of individual hard problems and corresponding algorithms. Here the researchers work mainly on the internal structures of different complexity theory classes and the relationships among them. The research in this direction was mainly started to solve P-NP problem. Most of the researches are done taking the assumption $\mathsf{P} \neq \mathsf{NP}$ and on a more far-reaching conjecture that the polynomial time hierarchy of complexity classes is infinite [27]. We already know that, if there exists certain compression algorithm, Polynomial Hierarchy collapses[32]. Basically, using the idea of compression, we can study the density or internal nature in the structure of different complexity classes. For example, Buhrman and Hitchcock [18] have shown that hard sets S for NP problems must have exponential density unless $\mathsf{CoNP} \subseteq \mathsf{NP} / poly$. These kind of results are very much related to the infeasibility of instance compression as one needs find compression algorithm (incompressibility proof) to prove such results.

1.3 Review of some important complexity theory notions

We are now going to discuss some of the existing and well known concepts in complexity theory which are very much related to our work.

1.3.1 Parameterized complexity and related notions

Parameterized complexity is very much related to our work. That is why we like to discuss some of the important concepts related to this domain in more details. We start with some basic idea behind this approach. The idea behind the field parameterized complexity, as mentioned in [41], is that we should look more deeply into the actual structure of the problem in order to seek some kind of hidden feasibility. It

can be considered as a sub-field in complexity theory, which is more fine-tuned to actual applications. The reason is, in actual application, very often we have some more knowledge about the problem structure. For example, in most of the graph-theoretic applications, the graph instance has some specific structure rather than just some general graph or in real life quantified Boolean circuit problems, we do not generally tend to work with more than a few alternations of quantifiers. Parameterized complexity theorists study these kind of problems extensively. We are now going to define some of the important concepts related to this field.

Definition 2. *[43] A parameterized problem is a set $L \subseteq \Sigma^* \times \Sigma^*$ where Σ is a fixed alphabet.*

We have already seen the definition of fixed parameter tractability in this context. But now let us see this definition in more formal way as given by Downey and Fellows [43].

Definition 3. *A parameterized problem L is (uniformly) fixed-parameter tractable if there exists a constant α and an algorithm to determine whether (x,y) is in L which runs in time $f(|y|).|x|^{\alpha}$, where $f : \mathbb{N} \to \mathbb{N}$ is an arbitrary function. The class of fixed-parameter tractable problems are denoted by FPT.*

There are parameterized problems which are not fixed-parameter tractable. So it is natural to ask if there is any way to classify these intractable problems with respect to degree of intractability. To answer this question, different hierarchies are developed in parameterized complexity domain. W-hierarchy ([43]) is the most popular among them. Before defining that hierarchy formally, we need to define some kind of reduction suitable for this domain. We are now going to see that definition. In this context we like to mention that, for a parameterized problem L and $y \in \Sigma^*$, L_y is generally used to denote the associated fixed-parameter problem (y is the parameter) $L_y = \{x|(x,y) \in L\}$.

Now, let us see the definition of the reduction from a parameterized problem L to another parameterized problem.

Definition 4. *[43] A (uniform) reduction of a parameterized problem L to a parameterized problem $L^{'}$ is an oracle algorithm A that on input (x,y) determines whether $x \in L_y$, and satisfies the following.*

1. *There is an arbitrary function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial q such that the running time of A is bounded by $f(|y|).q(|x|)$.*

2. *For each $y \in \Sigma^*$ there is a finite subset $J_y \subseteq \Sigma^*$ such that A consults oracles only for fixed-parameter decision problems L'_w where $w \in J_y$.*

If, additionally the functions f and $y \to J_y$ are both recursive we say that the reduction is strongly uniform. (All of the reductions in this paper are strongly uniform).

The implication of this reduction is better understood by the following result given in [43], lemma 1.1.

Lemma 1. *If the parameterized problem L reduces to the parameterized problem L', and if L' is fixed-parameter tractable, then L is also fixed-parameter tractable.*

We are now going to see the definition of W-hierarchy gradually. As mentioned above, W-hierarchy and VC-hierarchy ([13]) have some similarities. Both of them are defined using some circuit problems. We will discuss about this VC hierarchy in more details in our context in chapter 3. But before that we like to consider Boolean circuits.

Definition 5. *[43] A Boolean circuit is of mixed type if it consists of circuits having gates of the following kinds.*

1. ***Small gates:*** *Not gates, And gates and Or gates with bounded fan-in. It is usually assumed that the bound on fan-in is two for And gates and Or gates, and one for Not gates.*

2. ***Large gates:*** *AND gates and OR gates with unrestricted fan-in.*

Mixture of upper and lower cases is used to denote small gates (*And* gates and *Or* gates), and upper case to denote large gates (*AND* gates and *OR* gates).

Definition 6. *[43] The depth of a circuit C is defined to be the maximum number of gates (small or large), not counting not gates, on an input-output path in C. The weft of a circuit C is the maximum number of large gates on an input-output path in C.*

Using the similar notation as used in [43], we say that a family of circuits F has bounded depth if there is a constant h such that every circuit in the family F has depth at most h. We say that F has bounded weft if there is constant t such that every circuit in the family F has weft at most t. F is a decision circuit family if each circuit has a single output. A decision circuit C accepts an input vector x if the single output gate has value 1 on input x. The weight of a boolean vector x is the number of $1'$s in the vector. We have used same notation through out this work unless otherwise specified. Now we can understand the formal definition of W-hierarchy using the circuit problem defined next.

Definition 7. *[43] Let F be a family of decision circuits where F may have many different circuits with a given number of inputs. The parameterized circuit problem L_F is associated with F as follows, $L_F = \{(C,k) : C \in F$ and C accepts an input vector of weight $k\}$.*

Definition 8. *[43] A parameterized problem L belongs to $W[t]$ (monotone $W[t]$) if L uniformly reduces to the parameterized circuit problem $L_{F(t,h)}$ for the family $F(t,h)$ of mixed type (monotone) decision circuits of weft at most t, and depth at most h, for some constant h.*

We will see later that VC-hierarchy has some similarities with the above definition. Downey and Fellows have also shown the following relation among these classes in the W-hierarchy.

Theorem 1. *[42] $FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots$.*

We will now consider the example of some of the natural parameterized problems and their position in this hierarchy.

CLIQUE:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a clique of size k (a pairwise adjacent subset of k vertices).

Parameter: k

DOMINATING SET:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a dominating set of size k (a set D of k vertices for which every vertex not in D has a neighbour in D).

Parameter: k

These problems are quite popular and well known NP-complete problems. Among them CLIQUE is known to be $W[1]$-complete ([43]) and DOMINATING SET is known to be $W[2]$-complete [42].

If we consider the parameter of these problems, we can see that k is not actually the natural witness length for either of them. Hence, in our settings of instance compression, parameter choice should be different and we need different hierarchy with respect to compression. VC-hierarchy is defined in this context that we will discuss later (chapter 3).

Flum and Grohe have given an alternate characterization to define the same W-hierarchy ([25]). We discuss about that here as some of the hierarchies (WK and MK

hierarchy as in [14]) related to VC-hierarchy are defined in the similar way as characterized by Flum and Grohe. We will see those WK and MK hierarchies formally in chapter 3.

Flum and Grohe have termed the reduction between two parameterized problems as fpt-reduction which is defined as below. From the example above, we can see that the parameter is actually the function of that specific instance. This notion is used in the following definition.

Definition 9. *Let (Q,κ) and (Q',κ') be parameterized problems over the alphabets Σ and Σ', respectively. An fpt-reduction (more precisely, fpt many-one reduction) from (Q,κ) to (Q',κ') is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:*

1. *For all $x \in \Sigma^*$ we have $(x \in Q \leftrightarrow R(x) \in Q')$.*
2. *R is computable by an fpt-algorithm (with respect to κ). That is, there is a computable function f and a polynomial $p()$ such that $R(x)$ is computable in time $f(\kappa(x))p(|x|)$.*
3. *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'((R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.*

As mentioned above, they have also shown a different characterization of the W-hierarchy. Before that we need to consider some other definitions.

For $t \geq 0$ and $d \geq 1$, we inductively define the following classes $\Gamma_{t,d}$ and $\Delta_{t,d}$ of formulas following [25]:
$\Gamma_{0,d} := \{\lambda_1 \wedge \ldots \wedge \lambda_c : c \in [d]$ and $\lambda_1, \ldots, \lambda_c$ are literals $\}$,
$\Delta_{0,d} := \{\lambda_1 \vee \ldots \vee \lambda_c : c \in [d]$ and $\lambda_1, \ldots, \lambda_c$ are literals $\}$,
$\Gamma_{t+1,d} := \{\bigwedge_{i\in I} \delta_i$: I is a finite non-empty index set and $\delta_i \in \Delta_{t,d}$ for all $i \in I\}$,
$\Delta_{t+1,d} := \{\bigvee_{i\in I} \delta_i$: I is a finite non-empty index set and $\delta_i \in \Gamma_{t,d}$ for all $i \in I\}$.

Thus, it is easy to understand that $\Gamma_{1,3}$ is nothing but the set of all the 3-CNF formulae, and $\Gamma_{2,1}$ is nothing but the set of formulae in CNF.

Let us now consider any general class of formula Φ. We take Φ^+, $\Phi^- \subseteq \Phi$ to denote the restrictions of Φ to formulas containing only positive and negative literals, respectively. Now for any $t \geq 2$ we define the following problem.

Definition 10. *p-$WSat(\Gamma^+_{t,1})$: For any $t \geq 2$ consider the counting problem called p-$WSat(\Gamma^+_{t,1})$:*
***Input**: A formula ϕ over n variables of size m where $\phi \in \Gamma^+_{t,1}$ and an integer k ($\leq n$).*

***Parameter**: k*

***Task**: Decide whether there exists an assignments for ϕ such that exactly k variables are assigned to be True.*

In the similar way one can define $p\text{-}WSat(\Gamma^{-}_{t,1})$, $p\text{-}WSat(\Delta^{+}_{t,d})$ and $p\text{-}WSat(\Delta^{-}_{t,d})$ for $t \geq 2$ and $d \geq 1$. Now we can see that by the following theorem we can get a different characterization for the W-hierarchy (more detailed discussion about these concepts including the proofs can be found in [25]).

Theorem 2. *([25], Theorem 7.1) For every $t > 1$, the following problems are $W[t]$-complete under fpt-reductions:*

1. *$p\text{-}WSat(\Gamma^{+}_{t,1})$ if t is even and $p\text{-}WSat(\Gamma^{-}_{t,1})$ if t is odd.*
2. *$p\text{-}WSat(\Delta^{+}_{t+1,d})$ for every $d \geq 1$.*

Our work is very much related to this field of parameterized complexity theory as any compression algorithm (or incompressibility result) in our domain will contribute to the work related to polynomial kernelization. In many cases we have used some of the existing results or techniques to prove some of the results in the domain of instance compression. We will discuss about them in more details along with our results.

1.3.2 Counting problems and complexity class #P

So far we have seen those problems answer to which is either *yes* or *no*. These kind of problems are known as decision problems. Now we are going to see counting problems where rather than finding some witness for the *yes* instance, we are interested to know how many such witnesses are there. This concept was introduced by L. G. Valiant [33, 34]. He has defined the counting Turing machine as follows.

Definition 11. *A counting Turing machine is a standard non-deterministic Turing Machine with an auxiliary output device that (magically) prints in binary notation on a special tape the number of accepting computations induced by the input. It has (worst-case) time-complexity $f(n)$ if the longest accepting computation induced by the set of all inputs of size n takes $f(n)$ steps (when the TM is regarded as a standard non-deterministic machine with no auxiliary device).*

Depending on this, a new complexity class #P of counting problems are defined as follows.

Definition 12. *[33] #P is the class of functions that can be computed by counting TMs of polynomial time complexity.*

The hardness and completeness with respect to this class is also defined in [33] by *Oracle Turing Machine*. But these concepts are easier to understand in terms of witness function. For that we first see the definition of NP verification algorithm (taken from [40]).

Definition 13. *According to the definition of complexity class NP, for every language L in NP, there exists an algorithm V such that:*

1. *$x \in L$ if and only if there exists y such that $V(x,y) = 1$.*
2. *There exists a polynomial q such that if $V(x,y) = 1$ for some x and y then $|y| \leq q(|x|)$.*
3. *The running time of V is polynomial in its input.*

Here V is known as verification algorithm for L.

For the alphabet $\{0,1\}$, we now consider the function $w : \{0,1\}^* \to \wp(\{0,1\}^*)$, where $\wp(\{0,1\}^*)$ denotes the power set of $\{0,1\}^*$. If x is a problem instance, $w(x)$ is used to denote the witnesses of x. The function $G : \{0,1\}^* \to \mathbb{N}$ is termed as counting problem where for any problem instance x, $G(x) = |w(x)|$.

For satisfiability problem, suppose x is an encoding of the Boolean formula ϕ and $y \in \{0,1\}^*$ is an encoding of a Boolean assignment to its input variables. For this problem instance, set of witnesses is, $w(x) = \{y|V(x,y) = 1\}$, where V is a verification algorithm for satisfiability problem which verifies if y is a satisfiable assignment for ϕ or not.

In terms of witness function we can see the following definition of the class #P.

Definition 14. *[12] The class #P is the class of counting problems with witness functions w such that:*

1. *There is a polynomial-time algorithm to determine, for given x and y, if $y \in w(x)$.*
2. *There exists a constant $k \in \mathbb{N}$ such that for all $y \in w(x)$, $|y| \leq |x|^k$. (The constant k can depend on w).*

We are now going to see the definition of counting reductions which are used to define the concept of hardness and completeness.

Definition 15. *[12] Let $w : \{0,1\}^* \to \wp(\{0,1\}^*)$ and $v : \{0,1\}^* \to \wp(\{0,1\}^*)$ be the witness functions of two counting problems G and H respectively (where $\wp(\{0,1\}^*)$ denotes the power set of $\{0,1\}^*$). A weakly parsimonious reduction from G to H consists of a pair of polynomial-time computable functions $\sigma : \{0,1\}^* \to \{0,1\}^*$ and $\tau : \{0,1\}^* \times \mathbb{N} \to \mathbb{N}$ such that $|w(x)| = \tau(x, |v(\sigma(x))|)$.*

A counting reduction is strongly parsimonious if $|w(x)| = |v(\sigma(x))|$.

We can also see another notion of reduction.

Definition 16. *[12] Let $w : \{0,1\}^* \to \wp(\{0,1\}^*)$ and $v : \{0,1\}^* \to \wp(\{0,1\}^*)$ be the witness functions of two counting problems G and H respectively. A Turing-reduction from G to H is an algorithm with an oracle for $|v(y)|$ that finds $|w(x)|$ for any instance x of w in time polynomially bounded in $|x|$.*

It is easy to see that if there is a weakly parsimonious reduction from G to H, there is also a Turing-reduction from G to H. Now we can look into the following definition.

Definition 17. *A counting problem G is #P-hard if there is a Turing-reduction from H to G for all problems $H \in$ #P. A counting problem G is #P-complete if it is #P-hard and $G \in$ #P.*

There are many decision problems which are solvable in polynomial time where their counting versions are #P-hard. Permanent is probably the most popular among them. The Permanent problem for 0-1 matrices, which is equivalent to the problem of counting perfect matchings in a bipartite graph, is proved to be #P-complete by L. G. Valiant [33]. But the decision problem of finding a perfect matching in a bipartite graph is known to be in complexity class P ([22, 8, 35, 37, 20]). Not only Permanent, it can also be proved that many other easy P problems correspond to very hard #P problem when we consider corresponding counting counterpart. *DNF* formula satisfiability problem is one of them. We will use these concepts of counting problems later in chapter 4 in more details.

1.4 Contribution of this work

We are now going to give a brief overview regarding what we are going to see in the remaining part of the thesis. Main contribution of this project is divided into three chapters. Let us now see the motivation and importance of each of them individually.

1.4.1 Chapter 2: Instance Compression for the Polynomial Hierarchy and Beyond

In this chapter we have extended the concept of instance compression from its existing scope. The notion of instance compressibility for NP problems was defined by Harnik and Naor ([13]) which is closely related to the notion of polynomial kernelizability in the field of parametrized complexity. In this chapter, we define instance compressibility ([13, 17]) for parametric problems in higher complexity classes, PH and PSPACE. We observe that the parametric problem Σ_iCIRCUITSAT of deciding satisfiability of a quantified Boolean circuit with $i-1$ alternations of quantifiers starting with an existential quantifier, is complete for parametric problems in the class Σ_i^p with respect to w-reductions (definition is in chapter 2, it is the reduction from a parametric problem to another with respect to compression). Analogously it can be shown that the problem QBCSAT (Quantified Boolean Circuit Satisfiability) is complete for parametric problems in PSPACE with respect to w-reductions. We show the following results about these problems:

1. If CIRCUITSAT is non-uniformly compressible within NP, then Σ_iCIRCUITSAT is non-uniformly compressible within NP, for any $i \geq 1$.

2. If QBCSAT is non-uniformly compressible (or even if satisfiability of quantified Boolean CNF formulae is non-uniformly compressible), then PSPACE $\subseteq$ NP/poly and PH collapses to the third level.

Next, we define Succinct Interactive Proof (Succinct IP) and by adapting the proof of IP = PSPACE ([11, 6]), we show that QBCNFSAT (Quantified Boolean Formula (in CNF) Satisfiability) is in Succinct IP. This result is different compared to analogous result for Succinct PCP, as it is know that, if QBCNFSAT has Succinct PCPs ([32]), Polynomial Hierarchy (PH) collapses.

1.4.2 Chapter 3: VC hierarchy classification

One of the most important objectives of this research work is to provide a *structural theory* of compressibility, analogous to the theory in the classical settings of solvability. To extend this theory further after extending it to higher classes, the next step is to develop a hierarchical structure of the parametric problems with respect to compressibility, something similar to W-hierarchy. Such hierarchy is already defined (VC-hierarchy

[13]), but there are many open questions related to this. We have tried to answer some of the open questions in this phase and give some more interesting results.

Firstly, we have slightly modified the existing definitions of VC-hierarchy to make it suitable for parametric problems with natural witness length as parameter. The significance of this hierarchy is, the problems in the lower classes are comparatively easier to compress, and if we go higher, corresponding problems are tougher to compress. Although it was defined in [13], not too many natural problems were classified with respect to compression. Besides, there was not much significant hardness or completeness results for these classes in VC-hierarchy. In this chapter 3, we have considered a long list of natural NP problems and tried to classify them into some level of VC-hierarchy. We have shown some of the new w-reductions in this context and pointed out a few interesting results including the ones as follows.

1. CLIQUE is VC_1-complete (using the results in [14]).

2. SET SPLITTING and NAE-SAT are VC_2-complete.

Harnik and Naor have provided two ways to define this VC-hierarchy ([13]). We have considered both of them and studied them further for some other interesting parametric problems. We have also introduced a new class in this hierarchy and discussed its importance with some related results. We have done a comparison of this VC-hierarchy with other related hierarchies in parameterized complexity domain as well.

In this context we also like to mention that for any problem there can be more than one way to choose the actual parameter of the problem. The reason is, there can be different Non-deterministic Turing machine for the same problem and hence more than one way to choose the natural parameter. In our work we have considered same language with different parameter as different parametric problem and studied them separately to place them in VC-hierarchy.

1.4.3 Chapter 4: Counting hierarchy with respect to Compression

So far we have considered only the decision problems. We have already seen that the counting problems are difficult compared to decision problems and many easy decision problems have very hard counting counterpart. Hence it is a natural question to ask that how far the same thing is true with respect to compression. We have tried to answer that in this chapter 4 which helps to develop the structural theory of compressibility even further.

In this chapter we have defined parametric counting problems and a counting hierarchy depending on the notion of instance compression. We named this hierarchy as #VC-hierarchy. Similar approach to consider the counting problems is not used much in the field of parameterized complexity theory. Although, Downey and Fellows have defined the counting counterpart of W-hierarchy, i.e., #W-hierarchy [26] and presented some interesting results.

In this chapter we have discussed which of the reductions for decision problems are working for counting problems as well and which are not and analysed them in details to present analogous results in counting hierarchy. But we primarily focused on those counting problems which have different behaviour compared to their decision counter part. Other than that, we have considered some of the interesting practical problems here that we have not considered in the previous chapter and studied their complexity for both decision and counting version.

We have also considered a large variations of circuit satisfiability problems (e.g. weighted monotone satisfiability, exact satisfiability etc.) as these kind of problems are quite relevant to real world applications. It is because in real world problems, most often we have more information about the problem structure and/or we need to satisfy a few more extra conditions. That is why, we have studied them here in details to present some of the interesting results about those problems with respect to our theory of instance compressibility.

1.4.4 Chapter 5: Conclusion

In the final concluding chapter, we have mainly discussed some of the open questions that we could not answer. We have also given a brief overview of an alternate approach to consider counting hierarchy that we have defined in previous chapter. After that we have tried to make a list of interesting open problems and discuss the future direction of further research in this field. Here we have mainly discussed the future direction related to the work in chapter 3 and 4. For work related to chapter 2, open problems are discussed separately there as they are in slightly different direction.

Chapter 2

Instance Compression for the Polynomial Hierarchy and Beyond

An NP problem is said to be instance compressible if there is a polynomial-time reduction mapping instances of size m and parameter n to instances of size $poly(n)$ (possibly of a different problem). As mentioned already, the notion of instance compressibility for NP problems was defined by Harnik and Naor ([13]) motivated by applications in cryptography. This notion is closely related to the notion of polynomial kernelizability in parametrized complexity ([17, 47, 25]), which is motivated by algorithmic applications. Fortnow and Santhanam showed ([32], Theorem 3.1) that the compressibility of the satisfiability problem for Boolean formulae (even non-uniformly) is unlikely, since it would imply that the Polynomial Hierarchy (PH) collapses. Since then, there's been a very active stream of research extending this negative result to other problems in NP ([17, 55] etc.). Instance compressibility is a useful notion for complexity theory as well - Buhrman and Hitchcock [18] use it to study the question of whether NP has sub-exponentially-sparse complete sets.

Given different possibilities of application of this notion, it is a natural question whether we can extend it to other complexity classes, such as PH and PSPACE. Our first contribution here is to define such an extension. The key to defining instance compressibility for NP problems is that there is a notion of "witness" for instances of NP problems, and in general the witness size can be much smaller than the instance size. We observe that the characterization of PH and PSPACE using alternating time Turing machines yields a natural notion of "guess size" - namely the total number of non-deterministic or co-non-deterministic bits used during the computation. We use this characterization to extend the definition of compressibility to parametric problems

in PH and PSPACE in a natural way.

Some proposals ([24, 28]) have already been made in the parametrized complexity setting for defining in general the parametrized complexity analogue of a classical complexity class. Our definition (Section 2.1) seems similar in spirit, but all the problems we consider *are* in fact fixed-parameter tractable. What we are interested in is whether they are instance-compressible, or equivalently whether they have polynomial-size kernels.

One of our main motivations is to provide a structural theory of compressibility, analogous to the theory in the classical setting. Intuitively, instance compressibility provides a different, more relaxed notion of "solvability" than the traditional notion. So it is interesting to study what kinds of analogues to classical results hold for the new notion. The result of Fortnow and Santhanam ([32]) can be thought of as an analogue of the Karp-Lipton theorem ([46], Theorem 6.1), since non-uniform compressibility is a weakening of the notion of non-uniform solvability. Other well-known theorems in the classical setting are that NP has polynomial-size circuits iff all of PH does, as well as the Karp-Lipton theorem for PSPACE ([46], Theorem 4.1). The main results we prove here are analogues of these results for instance compressibility.

Our first main result is, if the language CIRCUITSAT is non-uniformly compressible within NP (i.e., the reduction is to an NP problem), then so is the language Σ_iCIRCUITSAT, which is in some sense complete for parametric problems in the class Σ_i^p. Note that we need a stronger assumption here compared to that in the Fortnow-Santhanam result ([32]): they need only to assume that SAT is compressible. This reflects the fact that the proof is more involved - it relies on the Fortnow-Santhanam result ([32]) as well as on the techniques used in the classical case. In addition, the code used by the hypothetical compression for CIRCUITSAT shows up not just in the resulting compression *algorithm* for Σ_iCIRCUITSAT, but also in the *instance* generated - this is why we need to work with circuits, as they can simulate any polynomial-time computation. This ability to interpret code as data is essential to our proof. We give more intuition about the proof in Section 2.2, where the detailed proof can also be found. We also observe that under the assumption of Σ_3CIRCUITSAT being compressible (we make no assumption about the complexity of the set we are reducing to, nor do we require the compression to be non-uniform), all of the PH is compressible as well.

Our second main result is that if QBCNFSAT is non-uniformly compressible, the Polynomial Hierarchy collapses to the third level. The proof of this is easier and is

a direct adaptation of the Fortnow-Santhanam technique ([32]) to PSPACE. Here we consider an "OR" version of the problem as they do, and derive the collapse of the hierarchy from the assumption that the OR version is compressible. In the case of NP, showing that compressing the OR version is at least as easy as compressing SAT is easier as there are no quantifiers; however, this is not the case for PSPACE and this is where we need to work a little harder.

Our third result is an analogue of the IP = PSPACE ([11, 6]) result in the parametric world. We define the class Succinct IP, which consists of parametric problems with interactive protocols where the total amount of communication is polynomial in the size of the parameter. We observe that the traditional proof of IP = PSPACE ([11, 6]) can be adapted to show that the problem of determining whether a quantified Boolean formula is valid, has succinct interactive proofs. This demonstrates a difference between succinctness in an interactive setting and succinctness in a non-interactive setting - it is shown in [32] that if SAT has succinct probabilistically checkable proofs, then PH collapses.

There are many open problems in the compressibility theory for NP, such as, whether there are any unlikely consequences of SAT being probabilistically compressible. Our hope is that extending the theory to larger classes such as PH and PSPACE will give us more "room" to work with. Besides, if we manage to settle these questions for the larger classes, the techniques can be translated back to NP.

2.1 Some Complexity Theory Notions

Definition 18. ***Parametric problem:*** *A parametric problem is a subset of* $\{ \langle x, 1^n \rangle \mid x \in \{0,1\}^*, n \in \mathbb{N} \}$. *The term n is known as the parameter of the problem.*

NP problems in parametric form: Now consider some popular NP languages in parametric form.

SAT = $\{\langle \varphi, 1^n \rangle \mid \varphi$ is a satisfiable formula in *CNF*, and n is the number of variables in $\varphi\}$.

VC = $\{\langle G, 1^{k\ log(m)} \rangle \mid G$ has a vertex cover of size at most k, where $m = |G|\}$.

CLIQUE = $\{\langle G, 1^{k\ log(m)} \rangle \mid G$ has a clique of size at least k, where $m = |G|\}$.

DOMINATINGSET = $\{\langle G, 1^{k\ log(m)} \rangle \mid G$ has a dominating set of size at most k, where $m = |G|\}$.

OR-SAT = $\{\langle \{\varphi_i\}, 1^n \rangle \mid$ At least one φ_i is satisfiable, and each φ_i is of bit-length at most $n\}$.

For the parametric problems above in NP, the parameter can be interpreted as the *witness size* for some natural *NTM* (Nondeterministic Turing Machine) deciding the language. For example in SAT, the number of variables, which captures the witness of satisfiability problem, is taken as the parameter. Note that in the definitions of the CLIQUE, VC and DOMINATINGSET problems, the parameter is $k\ log(m)$ rather than k as in the typical parametrized setting. This is because, here $k\ log(m)$ bits will be required to represent the witness. We say that a parametric problem is in NP if there is a polynomial-time *NTM* solving it. We are now formally going to define instance compression ([13, 32]) for parametric problems.

Definition 19. ***Compression of parametric problem:*** *Suppose here L is a parametric problem. L is said to be compressible within a complexity class A if there is a polynomial p(.), and a polynomial-time computable function* f*:* $\{0,1\}^* \to \{0,1\}^*$*, such that for each* $x \in \{0, 1\}^*$ *and* $n \in \mathbb{N}$*,* $|f(\langle x, 1^n\rangle)| \leq p(n)$ *and* $\langle x, 1^n\rangle \in L$ *iff* $f(\langle x, 1^n\rangle) \in L_A$ *for some problem* L_A *in the complexity class A.*

We say the parametric problem L is *compressible* in general, if there exists any such complexity class A as mentioned above, for the problem L to be compressed within. We can now define the probabilistic version of compression.

Definition 20. ***Probabilistic compression of parametric problem:*** *Let L be a parametric problem and* $A \subseteq \{0,1\}^*$*. L is said to be probabilistically compressible with error* $\varepsilon(t)$ *within A if there is a probabilistic polynomial-time computable function f such that for each* $x \in \{0,1\}^*$ *and* $n \in \mathbb{N}$*, with probability at least* $1-\varepsilon(|x|)$ *we have:*

1. $|f(\langle x, 1^n\rangle)| \leq poly(n)$
2. $f(\langle x, 1^n\rangle) \in A$ *iff* $x \in L$

L is probabilistically compressible if there is an A such that L is probabilistically compressible within A with error $1/3$*. L is errorless compressible if there is an A such that L is probabilistically compressible within A with error* 0*.*

Definition 21. ***Non-uniform Compression:*** *A parametric problem L is said to be compressible with advice if the compression function is computable in deterministic polynomial time when given access to an advice string of size s(m, n) which depends only on m and n but not on the actual instance. Here m is the length of the parametric problem instance and n is the parameter. L is non-uniformly compressible if s is polynomially bounded in m and n.*

In other words, we can say that the machine compressing the parametric problem in the preceding definition takes advice in case of *Non-uniform Compression*. We can think of advice string s as a Boolean circuit.

Definition 22. ***w-Reduction:*** *[13] Given parametric problems L_1 and L_2 , L_1 w-reduces to L_2 (denoted $L_1 \leq_w L_2$) if there is a polynomial-time computable function f and polynomials p_1 and p_2 such that:*

1. $f(\langle x, 1^{n_1} \rangle)$ *is of the form* $\langle y, 1^{n_2} \rangle$ *where* $n_2 \leq p_2(n_1)$.

2. $f(\langle x, 1^{n_1} \rangle) \in L_2$ *iff* $\langle x, 1^{n_1} \rangle \in L_1$.

The semantics of a w-reduction is that if L_1 w-reduces to L_2 , it is at least as hard to compress L_2 as it is to compress L_1 . If $L_1 \leq_w L_2$ and L_2 is compressible, then L_1 is compressible. One can prove that OR-SAT $\leq_w$ SAT ([57]). To prove this, we can introduce $log(m)$ new variables to encode which smaller formula in OR-SAT is staisfiable (m is number of sub-formulae in OR-SAT instance).

As we have already mentioned, our primary objective is to extend the idea of compression to higher classes, namely Polynomial Hierarchy (PH) and PSPACE (definitions can be found in standard complexity theory text book, e.g., [48]). In our work, by a quantified Boolean formula, we mean a Boolean formula in prenex normal form where the quantifiers are in the beginning as follows, $\psi = Q_1\ x_1\ Q_2\ x_2\ \ldots\ Q_n\ x_n\ \phi$, for any Boolean formula ϕ. Similarly we can consider quantified Boolean circuits. Let us now consider some standard PH and PSPACE languages but in parametric form.

CIRCUITSAT $= \{\langle C, 1^n \rangle \mid C$ *is a satisfiable circuit, and n is the number of variables in* $C\}$

Σ_iSAT $= \{\langle \varphi, 1^n \rangle \mid \varphi$ *is a satisfiable quantified Boolean formula in CNF (in prenex form) with* $i-1$ *alternations where odd position quantifiers are* $\exists$ *and even position quantifiers are* $\forall$*, and* $n = (n_1 + n_2 + \ldots + n_i)$ *where* n_i *is the number of the variables corresponding to* i^{th} *quantifier*$\}$

Σ_iCIRCUITSAT $= \{\langle C, 1^n \rangle \mid C$ *is a satisfiable quantified circuit (in prenex form) with* $i-1$ *alternations where odd position quantifiers are* $\exists$ *and even position quantifiers are* $\forall$*, and* $n = (n_1 + n_2 + \ldots + n_i)$ *where* n_i *is the number of the variables corresponding to* i^{th} *quantifier*$\}$

Similarly we can define Π_iSAT *and* Π_iCIRCUITSAT *in parametric form.*

QBCNFSAT $= \{\langle \varphi, 1^n \rangle \mid \varphi$ *is a satisfiable quantified Boolean formula (in prenex form) in CNF, and n is the number of variables*$\}$

QBFORMULASAT = $\{\langle\ \varphi,\ 1^n\ \rangle \mid \varphi$ *is a satisfiable quantified Boolean formula (in prenex form but not necessarily in CNF), and n is the number of variables*$\}$
If φ *is replaced by the circuit C, then similarly we can define* **QBCSAT**.

OR-QBCNFSAT = $\{\langle\ \{\varphi_i\ \},\ 1^n\ \rangle \mid$ *Each* φ_i *is a quantified Boolean formula in CNF (in prenex form) and at least one* φ_i *is satisfiable, and each* φ_i *is of bit-length at most n*$\}$.

Now we can generalize. For any language L we can define, **OR-L** = $\{\langle\ \{x_i\ \},\ 1^n\ \rangle$ | At least one $x_i \in$ L, and each x_i is of bit-length at most $n\}$.

Here we would like to mention that the non-parametric versions of Σ_iCIRCUITSAT and Σ_iSAT are complete for the class Σ_i^p according to Cook-Levin reduction, and similarly the non-parametric versions of QBCNFSAT, QBFORMULASAT and QBCSAT are complete for PSPACE.

We can define a parametric problem corresponding to any language L in the class Σ_i^p, or more precisely to the $(i+1)$-ary polynomial-time computable relation R defining L, as follows.

Definition 23. *For any* $(i+1)$*-ary polynomial-time computable relation R, we can define a parametric problem in* Σ_i^p, $L_R = \{\langle\ x,\ 1^n\ \rangle \mid \exists\ u_1 \in \{0,1\}^{n_1}\ \forall\ u_2 \in \{0,1\}^{n_2}$ $\ldots\ Q_i\ u_i \in \{0,1\}^{n_i}$ *R (x,* u_1 *, . . . ,* u_i*) = 1 and* $n = (n_1 + n_2 + \ldots + n_i)$ *where* n_i *is the parameter corresponding to* i^{th} *quantifier*$\}$

We can do essentially the similar thing for any language $L \in$ PSPACE using the characterization of PSPACE as alternating polynomial time ([3], Corollary 3.6) as follows:

Proposition 1. *Any language L is in* PSPACE *if and only if it is decidable by an Alternating Turing machine in polynomial time.*

Now we can define,

Definition 24. *For any binary polynomial-time computable relation R, we can define a parametric problem in* PSPACE, $L_R = \{\langle\ x,\ 1^n\ \rangle \mid Q_1\ u_1 \in \{0,1\}^{n_1}\ Q_2\ u_2 \in \{0,1\}^{n_2}$ $\ldots\ Q_i\ u_i \in \{0,1\}^{n_i}$ *R (x,* $\langle\ u_1\ ,\ \ldots\ ,\ u_i\ \rangle$*) = 1 and* $n = (n_1 + n_2 + \ldots + n_i)$ *where all the Q variables denote* $\exists$ *or* $\forall$ *alternately, depending on whether its suffix is odd or even, i is polynomially bounded with respect to the size of x,* $\langle\ \ldots\ \rangle$ *encodes a sequence of numbers as a bit string and* n_i *is the parameter corresponding to* i^{th} *quantifier*$\}$

So using the general definition of compression of any language in parametric form given above, we can define the compression for all the PH and PSPACE parametric problems where the "witness length" or "guess length" is the parameter of the problem.

Proposition 2. Σ_iCIRCUITSAT *is a complete parametric problem with respect to w-reduction for the class of parametric problems in* Σ_i^p.

Proof. Firstly we can observe that Σ_iCIRCUITSAT is among the parametric problems in the class Σ_i^p as there is an Alternating Turing Machine accepting this language with $i-1$ alternations, starting with existential guesses. Let us now consider the parametric problem $L_R \in \Sigma_i^p$. So there exists a polynomial-time computable relation R such that, $\langle x, 1^n \rangle \in L_R \Leftrightarrow \exists\, u_1 \in \{0,1\}^{n_1}\ \forall\, u_2 \in \{0,1\}^{n_2} \ldots\ Q_i\ u_i \in \{0,1\}^{n_i}\ R\,(x, u_1, \ldots, u_i) = 1$, where Q_i denotes $\exists$ or $\forall$ depending on whether i is odd or even respectively. Here $n = (n_1 + n_2 + \ldots + n_i)$.

Now for the parametric problem L_R the parameter is the number of guess bits used by R which is n in this case. We know that any polynomial time computable relation has uniform polynomial size circuits ([48], Theorem 6.7). Let C_m be the circuit on inputs of length m - we can generate C_m from 1^m in polynomial time. Hence, $\langle x, 1^n \rangle \in L_R \Leftrightarrow \exists\, u_1 \in \{0,1\}^{n_1}\ \forall\, u_2 \in \{0,1\}^{n_2} \ldots\ Q_i\ u_i \in \{0,1\}^{n_i}\ C_m\,(x, u_1, \ldots, u_i) = 1$, where Q_i denotes $\exists$ or $\forall$ depending on whether i is odd or even respectively. This gives a w-reduction from the parametric problem L_R to Σ_iCIRCUITSAT, since the length of the parameter is preserved. □

A similar proposition holds for Π_iCIRCUITSAT as well. We can also show, using a similar proof, a completeness result for PSPACE as follows.

Proposition 3. QBCSAT *is a complete parametric problem for the class of parametric problems in* PSPACE *with respect to w-reduction.*

We note that all the parametric problems we have defined so far are in fact fixed-parameter tractable, simply by using brute force search.

Proposition 4. QBCSAT *is solvable in time* $O(2^n poly(m))$ *by brute force enumeration.*

2.2 Instance Compression for Polynomial Hierarchy

2.2.1 Instance Compression in second level

In this section, we are going to show that non-uniform compression of CIRCUITSAT within NP implies the non-uniform compression of Σ_2CIRCUITSAT within NP as well. In the next subsection, essentially by using induction we show how to extend this to the entire Polynomial Hierarchy (our work related to these results is published in [9]). We have used the following result by Fortnow and Santhanam ([32], Theorem 3.1):

Theorem 3. *If* OR-SAT *is compressible, then* CoNP $\subseteq$ NP/poly*, and hence* PH *collapses.*

The same technique actually shows that, any language L for which OR-L (section 2.1) is compressible, lies within CoNP/poly.

Theorem 4. *If* CIRCUITSAT *is non-uniformly compressible within* NP*, then the problem* Σ_2CIRCUITSAT *is non-uniformly compressible within* NP.

We first give some intuition for the proof. The natural approach is to proceed as in the classical setting. Let us assume SAT is compressible, and suppose we wish to derive that Σ_2SAT is compressible. Let us consider an arbitrary Σ_2SAT instance ϕ of size m with n variables - let u be the existentially quantified variables, and v be the universally quantified ones. Suppose we fix u to say a string u_1. This gives an instance of a CoNP language whose input consists of ϕ together with u_1. We can use our compression hypothesis to compress this instance to a new instance of size polynomial in size of v, which is an instance of a CoNP language (since our original assumption is that NP problems are compressible within NP, it follows that CoNP problems are compressible within CoNP). However, what we are left with is *still* a Σ_2SAT formula.

This is where the result of Fortnow and Santhanam [32] comes useful. They show that under the compressibility assumption, NP $\subseteq$ CoNP/poly (as we know that CoNP $\subseteq$ NP/poly implies NP $\subseteq$ CoNP/poly), so the universal quantifier can be flipped (modulo a polynomial amount of advice). However, doing this still doesn't give us a Σ_1 formula of size polynomial in m. The reason is that the compressed formula we obtain when using the compressibility assumption *depends* on u_1. It is unclear how to get a single formula of size polynomial in m uniformly in u_1.

This is why we need to work with the assumption about CIRCUITSAT rather than about SAT. The idea is that since the compression operates in polynomial time, we can encode the output of the compression as a polynomial-size circuit. This allows us to get from our assumption a Σ_1 problem instance of size polynomial in m which is equivalent to our original instance - the new instance succinctly encodes compressed circuits for all possible u_1. Now we can use our compression assumption again to compress this Σ_1 problem instance and hence get a compressed form of our original Σ_2 problem instance as well. Throughout this procedure, we'll need to carry along various advice strings. The details are given below.

Proof. Let us consider the parametric problem Σ_2CIRCUITSAT first. For the sake of convenience, we often omit the parameter when talking about an instance of this problem. According to the definition,

$$C \in \Sigma_2\text{CIRCUITSAT} \Leftrightarrow \exists u \in \{0,1\}^{n_1} \forall v \in \{0,1\}^{n_2} C(u,v) = 1 \tag{2.1}$$

$$C \notin \Sigma_2\text{CIRCUITSAT} \Leftrightarrow \forall u \in \{0,1\}^{n_1} \exists v \in \{0,1\}^{n_2} C(u,v) = 0 \tag{2.2}$$

Let m be the length of the description of the circuit C and $n = (n_1 + n_2)$ to be the number of variables of C.

Let us now fix a specific $u = u_1$. Now, we can define a new parametric problem $L^{'}$ as follows,

$$\langle C, u_1, 1^{n_2} \rangle \in L^{'} \Leftrightarrow \forall v \in \{0,1\}^{n_2} C(u_1,v) = 1 \tag{2.3}$$

$$\langle C, u_1, 1^{n_2} \rangle \notin L^{'} \Leftrightarrow \exists v \in \{0,1\}^{n_2} C(u_1,v) = 0 \tag{2.4}$$

It is clear from the above definition that $L^{'}$ is a parametric problem in CoNP (of instance size $\leq O(m+n_1)$) and any instance of $L^{'}$ can be polynomial-time reduced to an instance of CIRCUIT-UNSAT, say $C^{'}$ (because CIRCUIT-UNSAT, the parametric problem of all unsatisfiable circuits, is CoNP-Complete with respect to w-reduction). As shown in Proposition 2, the size of the witness will be preserved in this reduction.

$C \in \Sigma_2\text{CIRCUITSAT} \Leftrightarrow \exists u_1 \langle C,\ u_1 \rangle \in L^{'}$ and $\langle C,\ u_1 \rangle \in L^{'} \Leftrightarrow C^{'} \in$ CIRCUIT-UNSAT. Here the instance length $|C| = m$ and $|C^{'}| = poly(m)$. $poly(.)$ is denoting just an arbitrary polynomial function.

Let g be the polynomial-time reduction used to obtain $C^{'}$ from C and u_1. Namely, $C^{'} = g(C,\ u_1)$. If CIRCUITSAT is non-uniformly compressible within NP, using the same reduction we can non-uniformly compress CIRCUIT-UNSAT within CoNP. That means we can reduce a CIRCUIT-UNSAT instance into another CIRCUIT-UNSAT instance in polynomial time, as CIRCUIT-UNSAT is CoNP-complete with respect to w-reduction. Assume this polynomial time compression function be f_1 with polynomial

size advice. So we will use f_1 to compress CIRCUIT-UNSAT instance $C^{'}$ to another CIRCUIT-UNSAT instance, say $C^{''}$, of size $poly(n_2)$.

Therefore, $C^{'} \in$ CIRCUIT-UNSAT $\Leftrightarrow C^{''} = f_1(\ C^{'},\ w_1\ ) = f_1(\ g(C,\ u_1),\ w_1\ ) \in$ CIRCUIT-UNSAT, where $|C^{''}| = poly(n_2)$ and the string w_1 (of size at most $poly(m)$) is capturing the notion of polynomial size advice. Here the compression function f_1 is computable in polynomial (in m) time.

Now, if CIRCUITSAT is non-uniformly compressible within NP so is SAT as SAT is w-reducible to CIRCUITSAT. Now, OR-SAT is also non-uniformly compressible as OR-SAT w-reduces to SAT. It can be proved from Theorem 3 that if OR-SAT is non-uniformly compressible then CoNP $\subseteq$ NP/poly, as mentioned in the beginning of this section.

Now combining the statements in the above paragraph we can say that if CIRCUIT-SAT is non-uniformly compressible within NP then CoNP $\subseteq$ NP/poly. So we can now reduce our parametric problem in CoNP (here CIRCUIT-UNSAT) instance $C^{''}$ to a NP-complete parametric problem instance using polynomial size advice. As CIRCUITSAT is a NP-complete with respect to w-reduction, we can reduce $C^{''}$ to a CIRCUITSAT instance, say $C^{'''}$, using a polynomial time computable function f_2 with advice w_2. In the above procedure, the length of the instance definitely will not increase by more than a polynomial factor. So clearly $|C^{'''}| = poly(n_2)$.

So from the above arguments we can say that,
$C^{'} \in$ CIRCUIT-UNSAT $\Leftrightarrow C^{'''} = f_2(\ C^{''},\ w_2\ ) = f_2(\ f_1(\ g(C,\ u_1),\ w_1\ ),\ w_2\ ) \in$ CIRCUIT-SAT, where $|C^{'''}| = poly(n_2)$ and the string w_2 (of size at most $poly(n_2)$) is capturing the notion of polynomial size advice which arises in the proof of Theorem 3. Here the function f_2 is computable in polynomial (in n_2) time.

Now we define a new circuit C_1 as follows. C_1 is a non-deterministic circuit whose non-deterministic input is divided into two strings: u of length n_1 and v of length $poly(n_2)$. Given its non-deterministic input, C_1 first computes $C^{'''} = f_2(\ (f_1(g(C,\ u),\ w_1),\ w_2)$. This can be done in polynomial size in m since the functions f_2, f_1 and g are all polynomial-time computable and C, w_1 and w_2 are all fixed strings of size polynomial in m. It then uses its input v as non-deterministic input to $C^{'''}$ and checks if v satisfies $C^{'''}$. This can be done in polynomial-size since the computation of a polynomial-size circuit can be simulated in polynomial time. If so, it outputs 1, else it outputs 0. Now we have

$$C \in \Sigma_2\text{CIRCUITSAT} \Leftrightarrow \exists u \in \{0,1\}^{n_1} \exists v \in \{0,1\}^{n_2} C_1(u,v) = 1 \tag{2.5}$$

$$C \notin \Sigma_2\text{CIRCUITSAT} \Leftrightarrow \forall u \in \{0,1\}^{n_1} \forall v \in \{0,1\}^{n_2} C_1(u,v) = 0 \tag{2.6}$$

The key point is that we have reduced our original Σ_2CIRCUITSAT question to a CIRCUITSAT question, *without* a super-polynomial blow-up in the witness size. This allows us to apply the compressibility hypothesis again. Also, note that C_1 is computable from C in polynomial size.

After that, using the compressibility assumption for CIRCUITSAT, we can non-uniformly compress C_1 to an instance C_2 of size $poly(n_1 + n_2)$ of a parametric problem in NP. Our final compression procedure just composes the procedures deriving C_1 from C and C_2 from C_1, and since each of these can be implemented in polynomial size, our compression of the original Σ_2CIRCUITSAT instance is a valid non-uniform instance compression. Thus it is shown that if CIRCUITSAT is non-uniformly compressible within NP, Σ_2CIRCUITSAT is also non-uniformly compressible within NP. □

Corollary 1. *If* CIRCUITSAT *is probabilistically compressible with error* $< 2^{-m}$ *within NP, where m is the instance size,* Σ_2CIRCUITSAT *is non-uniformly compressible within NP.*

Proof. This result is almost immediate from the above theorem. This extension uses "Adleman's trick" [29] to embed a "good" random string in the advice, and then applies the argument for non-uniform compression. Existence of such "good" random string can be seen by following argument. As the error is $< 2^{-m}$, by a union bound, there must be some choice of randomness that yields a correct compressed instance for each instance of length m. Let z be such a random string of size at most $poly(m)$ (since the compression algorithm runs in probabilistic polynomial time). Fortnow-Santhanam's result works for probabilistic compression with error $< 2^{-m}$ as well, because of same "Adleman's trick". Now we just use the same argument as in the proof above except that we also include z in the advice string. Hence, the corollary is proved. □

2.2.2 Instance Compression for higher levels in Polynomial Hierarchy

Now we are going to extend the idea for higher classes. It is not that difficult to see, if Σ_2CIRCUITSAT is non-uniformly compressible within NP, Π_2CIRCUITSAT is non-uniformly compressible within CoNP. We will use this in the following theorem.

Theorem 5. *If* CIRCUITSAT *is non-uniformly compressible within* NP*, then the problem* Σ_iCIRCUITSAT *is non-uniformly compressible within* NP *for all* $i > 1$.

Proof. Suppose C is a Σ_iCIRCUITSAT instance. So from the definition we can say that,

$C \in \Sigma_i$CIRCUITSAT $\Leftrightarrow \exists\, u_1 \in \{0,1\}^{n_1} \forall\, u_2 \in \{0,1\}^{n_2} \ldots Q_i\, u_i \in \{0,1\}^{n_i}\, C\,(u_1, \ldots, u_i) = 1$,

where Q_i denotes $\exists$ or $\forall$ depending on whether i is odd or even respectively.

Now, suppose CIRCUITSAT is compressible. To prove Σ_iCIRCUITSAT is compressible for all $i > 1$, we have to check the base case at the first place, that is for the case when $i = 2$. From the Theorem 1, we can say that if CIRCUITSAT is non-uniformly compressible within NP, Σ_2CIRCUITSAT is also non-uniformly compressible within NP. So the statement is true for base case.

Now suppose the statement is true for all $i \leq k$. We have to prove that the statement is true for $i = k + 1$ as well. So, assuming CIRCUITSAT is non-uniformly compressible within NP implies Σ_iCIRCUITSAT is non-uniformly compressible within NP for all $i \leq k$, we have to prove that Σ_{k+1}CIRCUITSAT is also non-uniformly compressible within NP.

Suppose C is a Σ_{k+1}CIRCUITSAT instance of size m. So from the definition we can say that,

$C \in \Sigma_{k+1}$CIRCUITSAT

$\Leftrightarrow \exists\, u_1 \in \{0,1\}^{n_1} \forall\, u_2 \in \{0,1\}^{n_2} \ldots Q_{k+1}\, u_{k+1} \in \{0,1\}^{n_{k+1}}\, C\,(u_1, \ldots, u_{k+1}) = 1$,

where Q_{k+1} denotes $\exists$ or $\forall$ depending on whether $(k+1)$ is odd or even respectively.

Now, let us fix u_1 to $u^{'}$. So now we can define a new parametric problem as follows,

$\langle C, u^{'}, 1^{n_2+n_3+\ldots+(n_{k+1})} \rangle \in L^{'} \Leftrightarrow \forall\, u_2 \in \{0,1\}^{n_2} \ldots Q_{k+1}\, u_{k+1} \in \{0,1\}^{n_{k+1}}\, C\,(u^{'}, u_2, \ldots, u_{k+1}) = 1$,

where Q_{k+1} denotes $\exists$ or $\forall$ depending on whether $(k+1)$ is odd or even respectively.

So it is clear from the above definition that $L^{'}$ is a parametric problem in Π_k^p (of instance size $\leq O(m + n_1)$) and any instance of $L^{'}$ can be polynomially reduced to an instance of Π_kCIRCUITSAT (because Π_kCIRCUITSAT is Π_k^p-Complete with respect to w-reduction). As shown in Proposition 2, the size of the witness will be preserved in this reduction. So this reduction is essentially a w-reduction. Suppose this Π_k^pCIRCUITSAT instance is $C^{'}$.

So from the above arguments,

$C \in \Sigma_{k+1}$CIRCUITSAT

$\Leftrightarrow \exists u^{'} \langle C, u^{'} \rangle \in L^{'}$ and $\langle C, u^{'} \rangle \in L^{'}$

$\Leftrightarrow C^{'} \in \Pi_k$CIRCUITSAT

Here the instance length $|C| = m$ and $|C^{'}| = poly(m)$. $poly(.)$ is denoting just an

arbitrary polynomial function.

Suppose g is the function to obtain $C^{'}$ from C, running in polynomial (in m) time. Namely, $C^{'} = g(C, u^{'})$.

From the induction hypothesis we can say, Σ_kCIRCUITSAT is non-uniformly compressible within NP. So any Π_kCIRCUITSAT instance, say $C^{'}$ is non-uniformly compressible to a CoNP instance as Π_kCIRCUITSAT = CoΣ_kCIRCUITSAT. After compression suppose the instance is $C^{''}$ which, without loss of generality, can be taken as a CIRCUIT-UNSAT instance as it is a complete for CoNP with respect to w-reduction. Here $|C^{''}| = poly(n^{'})$ where $n^{'} = (n_2 + n_3 + \ldots + n_{k+1})$
So, $C^{'} \in \Pi_k$CIRCUITSAT $\Leftrightarrow C^{''} \in$ CIRCUIT-UNSAT.

Assume f_1 to be the above compression function. So from the above arguments we can say,
$C^{'} \in \Pi_k$CIRCUITSAT $\Leftrightarrow C^{''} = f_1(\, C^{'}, w_1\,) = f_1(\, g(C, u^{'}), w_1\,) \in$ CIRCUIT-UNSAT, where $|C^{''}| = poly(n^{'})$ and the string w_1 (of size at most $poly(m)$) is capturing the notion of polynomial size advice. Here the compression function f_1 is running in polynomial (in m) time.

Now, if CIRCUITSAT is non-uniformly compressible within NP so is SAT as SAT is w-reduced to CIRCUITSAT. Now, OR-SAT is also non-uniformly compressible as OR-SAT w-reduces to SAT. It can be proved from Theorem 3 that if OR-SAT is non-uniformly compressible then CoNP $\subseteq$ NP/poly, as mentioned in the beginning of this section.

Now combining the above statements we can say that if CIRCUITSAT is non-uniformly compressible within NP then CoNP $\subseteq$ NP/poly. So we can now reduce our instance of parametric problem in CoNP (here CIRCUIT-UNSAT), $C^{''}$ into an instance of a parametric problem which is NP-complete, using polynomial size advice. As CIRCUITSAT is a NP-complete with respect to w-reduction, we can reduce $C^{''}$ to a CIRCUITSAT instance, say $C^{'''}$, using a polynomial time computable function f_2 with advice w_2. In the above procedure, the length of the instance definitely will not increase by more than a polynomial factor. So clearly $|C^{'''}| = poly(n_{\prime})$.

So from the above arguments we can say that,
$C^{'} \in \Pi_k$CIRCUITSAT $\Leftrightarrow C^{'''} = f_2(\, C^{''}, w_2\,) = f_2(\, f_1(\, g(C, u^{'}), w_1\,), w_2\,) \in$ CIRCUIT-SAT, where $|C^{'''}| = poly(n^{'})$ and the string w_2 (of size at most $poly(n^{'})$) is capturing the notion of polynomial size advice which arises in the proof of Theorem 3. Here the compression function f_2 is running in polynomial (in $n^{'}$) time.

Now we define a new circuit C_1 as follows. C_1 is a non-deterministic circuit whose

non-deterministic input is divided into two strings: u_1 of length n_1 and v of length $poly(n^{'})$. Given its non-deterministic input, C_1 first computes $C^{'''} = f_2(\ (f_1(g(C,\ u_1),\ w_1),\ w_2)$. This can be done in polynomial size in m since the functions f_2, f_1 and g are all polynomial-time computable and C, w_1 and w_2 are all fixed strings of size polynomial in m. It then uses its input v as non-deterministic input to C''' and checks if v satisfies C'''. This can be done in polynomial-size since the computation of a polynomial-size circuit can be simulated in polynomial time. If so, it outputs 1, else it outputs 0.

Now we have,

$$C \in \Sigma_{k+1}\text{CIRCUITSAT} \Leftrightarrow \exists u_1 \in \{0,1\}^{n_1}\ \exists v \in \{0,1\}^{n^{'}} C_1\ (u_1, v) = 1 \tag{2.7}$$

$$C \notin \Sigma_{k+1}\text{CIRCUITSAT} \Leftrightarrow \forall u_1 \in \{0,1\}^{n_1}\ \forall v \in \{0,1\}^{n^{'}} C_1\ (u_1, v) = 0 \tag{2.8}$$

The key point is that we have reduced our original Σ_{k+1}CIRCUITSAT question to a CIRCUITSAT question, *without* a super-polynomial blow-up in the witness size. This allows us to apply the compressibility hypothesis again. Also, note that C_1 is computable from C in polynomial size.

After that, using the compressibility assumption for CIRCUITSAT, we can non-uniformly compress C_1 to an instance C_2 of size $poly(n_1 + n^{'})$ i.e., $poly(n_1 + n_2 + \ldots + n_{k+1})$ of a parametric problem in NP. Our final compression procedure just composes the procedures deriving C_1 from C and C_2 from C_1, and since each of these can be implemented in polynomial size, our compression of the original Σ_{k+1}CIRCUITSAT instance is a valid non-uniform instance compression.

So using mathematical induction we can say if CIRCUITSAT is non-uniformly compressible within NP, Σ_iCIRCUITSAT is also non-uniformly compressible within NP for all $i > 1$. □

Corollary 2. *If* CIRCUITSAT *is compressible within* NP, Π_iCIRCUITSAT *is also non-uniformly compressible within* NP *for all* $i \geq 1$.

As Π_iCIRCUITSAT *w*-reduces to Σ_{i+1}CIRCUITSAT, the above Corollary is trivial. Theorems 4 and 5 require an assumption on non-uniform compressibility in NP. But we don't need this for compressibility of a problem higher in the hierarchy.

Proposition 5. *If* Σ_3CIRCUITSAT *is compressible, then* Σ_iCIRCUITSAT *is compressible for any* $i > 3$.

Proof. This proposition follows from the fact that Σ_3CIRCUITSAT being compressible implies that SAT is compressible. So, by the result of Fortnow and Santhanam (Theorem 3), PH collapses to Σ_3^p. As a result, every parametric problem in the class Σ_i^p w-reduces to Σ_3CIRCUITSAT, as Σ_3CIRCUITSAT is complete for the class Σ_3^p with respect to w-reduction. Hence, Σ_3CIRCUITSAT being compressible, Σ_iCIRCUITSAT is compressible for any $i > 3$. □

We can also find the following result easily using the "Adleman's trick" as mentioned already.

Corollary 3. *If* CIRCUITSAT *is probabilistically compressible with error* $< 2^{-m}$ *within NP, where m is the instance size,* Σ_iCIRCUITSAT *is non-uniformly compressible within NP for all* $i > 1$.

2.3 Instance Compression for PSPACE

In this section, we show that QBCNFSAT is unlikely to be compressible, even non-uniformly - compressibility of QBCNFSAT implies that PSPACE collapses to the third level of the Polynomial Hierarchy. The strategy we adopt is similar to that in Theorem 3 where it is shown that compressibility of SAT implies NP $\subseteq$ CoNP/poly. To show their result, they used the OR-SAT problem, which is w-reducible to SAT ([57]). Thus an incompressibility result for the OR-SAT problem translates directly to a corresponding result for SAT.

We similarly defined OR-QBCNFSAT problem in Section 2.1. But it is not that easy to show that OR-QBCNFSAT w-reduces to QBCNFSAT. There are a couple of different issues. First the quantifier patterns for the formulae $\{\phi_i\}, i = 1 \ldots m$ might all be different. This is easily taken care of, because we can assume quantifiers alternate between existential and universal - this just blows up the number of variables for any formula by a factor of at most 2. The more critical issue is that nothing as simple as the OR works for combining formulae. $\exists x \forall y \phi_1(x,y) \vee \exists x \forall y \phi_2(x,y)$ is not equivalent to $\exists x \forall y (\phi_1(x,y) \vee \phi_2(x,y))$. We are forced to adopt a different strategy as explained below. Later we have found similar strategy is used in [57], though it was in the context of OR-SAT, not OR-QBCNFSAT.

Lemma 2. OR-QBCNFSAT *is w-reducible to* QBCNFSAT

Proof. Let $\langle\{\phi_i\}, 1^n\rangle$ be an OR-QBCNFSAT instance of length m. Assume without loss of generality that each ϕ_i has exactly n variables and that the quantifiers alternate starting with existential quantification over x_1, continuing with quantification over x_2, x_3 etc. We construct in polynomial time in m an equivalent instance of QBCNFSAT with at most $poly(n)$ variables and of size $poly(m)$. We first take care of quantifications. The quantifier patterns for the formulae $\{\phi_i\}, i = 1 \ldots m$ might all be different. But we can assume quantifiers alternate between existential and universal - this just blows up the number of variables for any formula by a factor of at most 2. Then we check if the number of input formulae is greater than 2^n or not. If yes, we solve the original instance by brute force search and output either a trivial true formula or a trivial false formula depending on the result of the search. If not, then we define a new formula with $\lceil log(m) \rceil$ additional variables $y_1, y_2 \ldots y_k$. We identify each number between 1 and m uniquely with a string in $\{0,1\}^k$. Now we define the formula ψ_i corresponding to ϕ_i as follows. Let the string $w_i \in \{0,1\}^k$ correspond to the number i. Then $\psi_i = z_1 \wedge z_2 \ldots \wedge z_k \wedge \phi_i$, where $z_r = y_r$ if $w_r = 1$ and the complement of y_r otherwise. The output formula ψ starts with existential quantification over the y variables followed by the standard pattern of quantification over the x variables followed by the formula which is the OR of all ψ_i's, $i = 1 \ldots m$. So ψ will be as follows:
$\psi = \exists\, y_1\ \exists\, y_2 \ldots \exists\, y_k\ Q_1\, x_1\ Q_2\, x_2 \ldots Q_n\, x_n\ (\psi_1 \vee \psi_2 \vee \ldots \vee \psi_m)$.
Where Q_i's are the quantifications of the x_i's as before. It is not that hard to check that ψ is valid iff one of the ϕ_i's is. □

We are now going to show a result related to the notion of incompressibility of QBCNFSAT problem (our work related to this result is published in [9])

Theorem 6. *If* QBCNFSAT *is compressible, then* PSPACE $\subseteq$ NP/poly*, and hence* PSPACE = Σ_3^p.

Proof. Using Lemma 2, if QBCNFSAT is compressible, OR-QBCNFSAT is also compressible. From the proof of Theorem 3 we can say for any parametric problem L for which OR-L (section 2.1) is compressible, lies in CoNP/poly. Thus, since the parametric problem QBCNFSAT is PSPACE-complete and PSPACE is closed under complementation, a compression for OR-QBCNFSAT implies PSPACE is in NP/poly. Hence by the result of Yap [10], it follows that PH collapses to the third level. Combining this with the Karp-Lipton theorem for PSPACE ([46], Theorem 4.1), we have that PSPACE = Σ_3^p. □

In the similar manner, we can consider a small amount of error here which doesn't make any difference in the result.

Corollary 4. *If* QBCNFSAT *is probabilistically compressible with error* $< 2^{-m}$ *within NP, where m is the instance size, PSPACE* $= \Sigma_3^p$.

2.4 Succinct IP and PSPACE

IP ([54, 30]) is the class of problems solvable by an interactive proof system. An interactive proof system consists of two machines, a *Prover*, P, which presents a proof that a input string is a member of some language, and a $Verifier$, V, that checks that the presented proof is correct. Now we are extending this idea of IP to Succinct IP, where the total number of bits communicated between *prover* and the $verifier$ in all the interactions is polynomially bounded in parameter length instead of input instance size.

We define $Verifier$ to be a function V that computes its next transmission to the *Prover* from the message history sent so far. The function V has three inputs:
(1) **Input String**, (2) **Random bits** and (3) **Partial message history**

$m_1\#m_2\#\ldots\#m_i$ is used to represent the exchange of messages m_1 through m_i between P and V. The Verifier's output is either the next message m_{i+1} in the sequence or *accept* or *reject*, designating the conclusion of the interaction. Thus V has the function from V: $\Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \cup \{$ accept, reject $\}$.
The *Prover* is a party with unlimited computational ability. We define it to be a function P with two inputs:
(1) **Input String** and (2) **Partial message history**
The Prover's output is the next message to the Verifier. Formally, P: $\Sigma^* \times \Sigma^* \to \Sigma^*$.
Next we define the interaction between Prover and the Verifier. For particular input string w and random string r, we write $(V \leftrightarrow P)(w, r) = accept$ if a message sequence m_1 to m_k exists for some k whereby

1. for $0 \leq i < k$, where i is an even number, $V(w, r, m_1\#m_2\#\ldots\#m_i) = m_{i+1}$;
2. $0 < i < k$, where i is an odd number, $P(w, m_1\#m_2\#\ldots\#m_i) = m_{i+1}$; and
3. the final message m_k in the message history is $accept$.

In the definition of the class Succinct IP, the lengths of the Verifier's random bits and each of the messages exchanged are $p(n)$ for some polynomial p that depends only

on the Verifier. Here n is the parameter length of input instance. Besides, total bits of messages exchanged is at most $p(n)$ as well.

Succinct IP: A parametric problem L ($\subseteq \{\langle x, 1^n\rangle | x \in \{0,1\}^*, n \in \mathbb{N}\}$) is in Succinct IP if there exist some polynomial time function V and arbitrary function P, with total $poly(n)$ many bits of messages communicated between them and for every function $\tilde{P}$ and string w,

1. $w \in L$ implies $\Pr[V \leftrightarrow P] \geq 2/3$, and
2. $w \notin L$ implies $\Pr[V \leftrightarrow \tilde{P}] \leq 1/3$.

Here $poly(n)$ denotes some polynomial that depends only on the Verifier and n is the parameter length of input instance w.

We know that QBCNFSAT is in IP, as IP = PSPACE ([11, 6]). But we can even prove something more. We can actually construct Succinct IP protocol for QBCNFSAT and hence claim that QBCNFSAT $\in$ SUCCINCT IP. To prove that we are basically going to scrutinize the existing protocol in the formal proof of the part, PSPACE $\subseteq$ IP ([11, 6, 39]) and establish the succinctness result.

Theorem 7. QBCNFSAT $\in$ SUCCINCT IP

Proof. The key idea is to take an algebraic view of Boolean formulae by representing them as polynomials. We are considering the inputs are from some finite field $\mathbb{F}$. We can see that 0, 1 can be thought of both as truth values and as elements of $\mathbb{F}$. Thus we have the following correspondence between formulae and polynomials when the variables take 0/1 values:

$x \wedge y \leftrightarrow X \,.\, Y$

$\bar{x} \leftrightarrow 1$ - X

$x \vee y \leftrightarrow X$+$Y$

This is termed as arithmetization of Boolean formula. We can see that if a Boolean expression is satisfiable for certain Boolean assignment, value of the polynomial corresponding to that will be non-zero. So, if there is a Boolean formula $\phi(x_1, x_2, \ldots, x_n)$ of length m in CNF, we can easily convert that into a polynomial p of degree at most m following the rules described above. But we can observe that if $\phi(x_1, x_2, \ldots, x_n)$ is in DNF (Disjunctive Normal Form), the degree of any single variable will be at most 1, which may not be the case if ϕ is in CNF. So in this proof technique, we will convert a formula in CNF to DNF, to ensure that the degree of the polynomial is not too large.

Let the given formula be,

$\Upsilon = q_1\ x_1\ q_2\ x_2\ q_3\ x_3 \ldots\ q_n\ x_n\ \phi(x_1, \ldots, x_n)$,

where the size of Υ is m. ϕ is any Boolean formula over n variables in CNF. Here $q_i \in \{\forall, \exists\}$, for all $i = 1, 2, \ldots, n$.

So we need to arithmetize the quantifiers as well to arithmetize Υ. We will replace $\exists\ y$ by $\sum_{Y\in\{0,1\}}$ and $\forall\ y$ by $\prod_{Y\in\{0,1\}}$ and then arithmetize rest of the Boolean expression as described above. After this arithmetization of quantifiers, we can evaluate the polynomial to some integer value. So corresponding to any quantified Boolean formula we can get some numeric value (in the finite field $\mathbb{F}$ which will be chosen later). It can be proved easily by mathematical induction that a quantified Boolean formula is satisfiable iff corresponding numeric value after arithmetization will be non-zero [6].

Let us now consider negation of Υ as follows. $\Psi = \bar{\Upsilon} = Q_1\ x_1\ Q_2\ x_2\ Q_3\ x_3 \ldots\ Q_n\ x_n\ \bar{\phi}(x_1, \ldots, x_n)$ where Q_i is $\exists$ or $\forall$ if q_i is $\forall$ and $\exists$ respectively, for all $i = 1, 2, \ldots, n$. If ϕ is in CNF, clearly $\bar{\phi}$ will be in DNF. Hence the polynomial corresponding to $\bar{\phi}$ will have degree at most 1 for any single variable, as discussed above. We can say that if Υ is satisfiable, Ψ must be un-satisfiable, and hence corresponding numeric value for Ψ after arithmetization will be 0.

To arithmetize Ψ we will convert $\forall$ to multiplication and $\exists$ to summation as described below. But this conversion can once again increase the degree of the overall polynomial which was quite smaller before for un-quantified formula $\bar{\phi}$. To reduce that degree, we will now introduce some new terms in quantification and rewrite the expression in the following manner:

$\Psi' = Q_1\ x_1\ R\ x_1\ Q_2\ x_2\ R\ x_1\ R\ x_2\ Q_3\ x_3\ R\ x_1\ R\ x_2\ R\ x_3 \ldots\ Q_{n-1}\ x_{n-1}\ R\ x_1\ R\ x_2\ \ldots R\ x_{n-1}\ Q_n\ x_n\ \bar{\phi}(x_1, \ldots, x_n)$,

Here R is introduced to enable linearise operation on the polynomial as explained later. We now rewrite this Ψ' as follows : $\Psi' = S_1\ x_1\ S_2\ x_2\ S_3\ x_3 \ldots\ S_k\ x_k\ [\bar{\phi}]$,

where each $S_i \in \{\exists, \forall, R\}$. We are going to define R very soon. We can see that value of k can be at most $O(n^2)$.

For each $i \leq k$ we define the function f_i. We define $f_k(x_1, x_2, \ldots, x_n)$ to be the polynomial p [i.e., $p(x_1, x_2, \ldots, x_n)$] obtained by arithmetization of $\bar{\phi}$. For $i < k$ we define f_i in terms of f_{i+1}:

$S_{i+1} = \forall$: $f_i(\ldots) = f_{i+1}(\ldots, 0).f_{i+1}(\ldots, 1)$;

$S_{i+1} = \exists$: $f_i(\ldots) = f_{i+1}(\ldots, 0) + f_{i+1}(\ldots, 1)$;

$S_{i+1} = R$: $f_i(\ldots, a) = (1\text{-}a)f_{i+1}(\ldots, 0) + a f_{i+1}(\ldots, 1)$.

Here we reorder the inputs of the functions in such a way that variable y_{i+1} is

always the last argument. If S is $\exists$ or $\forall$, f_i has one fewer input variable than f_{i+1} does. But if S is R, both of them have same number of arguments. To avoid complicated subscripts, we use "…" which can be replaced by a_1 through a_j for appropriate values of j after the reordering of the inputs.

We can observe that operation R on polynomial doesn't change their values for Boolean inputs. So $f_0()$ is still the numeric value of Ψ' after arithmetization. Now we can observe that these Rx operation produces a result that is linear in x. We added $Rx_1\ Rx_2\ \ldots Rx_i$ after $Q_i x_i$ in Ψ' in order to reduce the degree of each variable to 1 prior to the squaring due to arithmetization of Q_i.

We are now ready to describe the protocol. P is denoted to be the prover and V to be the verifier as we always use.

Phase 0: [P sends $f_0()$]

$P \to V$: P sends $f_0()$ to V. V checks that $f_0() = 0$ and *rejects* if not.

Progressing similarly,

Phase i: [P persuades V that $f_{i-1}(r_1,\ldots)$ is correct if $f_i(r_1,\ldots,r)$ is correct]

$P \to V$: P sends the coefficients of $f_i(r_1,\ldots,z)$ as a polynomial in z. (Here r_1 … denotes a setting of the variables to the previously selected random values r_1, r_2, …)

V uses these coefficients to evaluate $f_i(r_1,\ldots,0)$ and $f_i(r_1,\ldots,1)$. Then it checks that the polynomial degree is at most 2 and that these identities hold:

$$f_{i-1}(r_1,\ldots) = \begin{cases} f_i(r_1,\ldots,0).f_i(r_1,\ldots,1) & \text{if } S_i = \forall \\ f_i(r_1,\ldots,0) + f_i(r_1,\ldots,1) & \text{if } S_i = \exists \end{cases}$$

and

$$f_{i-1}(r_1,\ldots,r) = (1-r)f_i(r_1,\ldots,0) + rf_i(r_1,\ldots,1) \text{ if } S_i = R$$

If either fails, V *rejects*.

$V \to P$: V picks a random Boolean value r from $\mathbb{F}$ and sends it to P. If $S_i = R$, this r replaces the previous r

Then it goes to phase i+1, where P must persuade V that $f_i(r_1,\ldots,r)$ is correct.

Progressing similarly,

Phase k**+1**: [V checks directly that $f_k(r_1,\ldots,r_n)$ is correct]

V evaluates $p(r_1,\ldots,r_n)$ to compare with the value V has for $f_k(r_1,\ldots,r_n)$. If they are equal, V *accepts*, otherwise V *rejects*. That completes the description of the protocol.

Here polynomial p is nothing but the arithmetization of the formula $\bar{\phi}$, as we have already seen. It can be shown that the evaluation of this polynomial can be done in polynomial time.

For the evaluation of the polynomial p for $r_1, \ldots, r_n$, we will consider $\bar{\phi}$ and apply the arithmetization for the nodes individually. We will evaluate the nodes from lower level. Before we evaluate for any node, corresponding inputs are already evaluated and ready to use. Evaluation for each node will take constant amount of time. So total evaluation of p for $r_1, \ldots, r_n$ through modified $\bar{\phi}$ will take $poly(m)$ time.

Now we can try to prove that the probability of error is bounded within the limit. If the prover P always returns the correct polynomial, it will always convince V . If P is not honest then we are going to prove that V rejects with high probability:

$$Pr[V\ rejects] \geq (1 - d/|\mathbb{F}|)^k \tag{2.9}$$

where d is the highest degree of the polynomial sent in each stage. We can see that value of k can be at most $O(n^2)$. As the value of d is 2 in our case, the right hand side of the above expression is at least (1 - $2k/|\mathbb{F}|$), which is very close to 1 for sufficiently large values of $|\mathbb{F}|$. It will be sufficient for us if $|\mathbb{F}|$ is bounded by a large enough polynomial in m not dividing the numeric value of Ψ' when it is not 0. Using the prime number theorem we know that we can find such value with small error bound as there are much more prime numbers $\leq poly(m)$ than number of prime factors of the numeric value of Ψ' (can be found in standard number theory book, e.g., [16]).

Now we are going to see how the proof works when the prover is trying to cheat for "no" instance. In the first round, the prover P should send $f_0()$ which must be 0. Then P is supposed to return the polynomial f_1. If it indeed returns f_1 then since $f_1(0)$ + $f_1(1) \neq f_0()$ by assumption, V will immediately reject (i.e., with probability 1). So assume that the prover returns some $s(X_1)$, different from $f_1(X_1)$. Since the degree d non-zero polynomial $s(X_1)$ - $f_1(X_1)$ has at most d roots, there are at most d values r such that $s(r) = f_1(r)$. Thus when V picks a random r,

$$Pr_r[s(r)\ \neq\ f_1(r)]\ \geq\ (1 - d/|\mathbb{F}|) \tag{2.10}$$

Then the prover is left with an incorrect claim to prove in all the phases. So prover should lie continuously. If P is lucky, V will not understand the lie. To prove equation (2.9), we will use induction here. We assume the induction hypothesis to be true for $k-1$ steps, that is, the prover fails to prove this false claim with probability at least $\geq (1 - d/|\mathbb{F}|)^{k-1}$. Base case is easy to see from equation (2.10). Thus we have,

$$Pr[V\ rejects]\ \geq\ (1 - d/|\mathbb{F}|).(1 - d/|\mathbb{F}|)^{k-1}\ = (1 - d/|\mathbb{F}|)^k \tag{2.11}$$

If P is not lucky, as the verifier is evaluating $p()$ explicitly in the last stage, V will anyway detect the lie.

Here in the description of the protocol, we can see that the degree of the polynomial at each stage is at most 2. So we need just constant number of coefficients for encoding such polynomials. Coefficients are from the field $\mathbb{F}$ which is of size $poly(m)$. So $O(log(poly(m)))$ i.e., $O(poly(n))$ size messages are sent in any phase. Even, it will be sufficient for us if $|\mathbb{F}|$ is bounded by a large enough polynomial in n. Number of such phases are bounded by (k+1) which is $O(n^2)$. So we have constructed a *Succinct* Interactive proof protocol for QBCNFSAT. □

Issue in finding Succinct IP protocol for QBCSAT: In case of QBCSAT, similar arithmetization technique will give polynomial of degree much larger size, actually exponential in m. Now, to reduce the error, we have to use Field $\mathbb{F}$ of larger size, basically exponential in m. This will give us each coefficients of the polynomials exchanged between *prover* and $verifier$ to be of size $log(e^{poly(m)})$, i.e., $poly(m)$, which means the protocol is not succinct.

2.5 Related Open Questions

There are various possible future directions from this work. Suppose CIRCUITSAT is compressible within a class C. In this work we have considered C to be the class NP and got some interesting results. For any general class C we know from Theorem 3 that the immediate consequence is the collapse of PH at third level. But it is still not known how our results for compression at second level of Polynomial Hierarchy will be affected for compression into an arbitrary class C. Besides, one could try to work under the weaker assumption that SAT or OR-SAT or OR-CIRCUITSAT is compressible instead of CIRCUITSAT. We also don't know whether there are similar implications for probabilistic compression where we allow certain amount of error in compression. One could also try to find a Succinct IP protocol for QBCSAT to show Succinct IP = PSPACE or try to find some negative implications of such a protocol existing for QBCSAT.

Chapter 3

VC hierarchy classification

One of the most important objectives of this research work is to provide a *structural theory* of compressibility, analogous to the theory in the classical settings of tractability. We have already defined compressibility for higher complexity classes in the previous chapter and obtained both positive and negative results. So the next natural question that arises in our mind is whether there is any way to classify these problems further with respect to compressibility. Hence, to extend the development of theory of compressibility, the next step is to develop a hierarchy of the parametric problems depending on compressibility, something similar to W-hierarchy. Such hierarchy is already defined (VC-hierarchy [13]), but there are many open questions related to this. We have tried to answer some of the open questions in this chapter and give some more interesting results.

We have introduced a new class in the VC-hierarchy and discussed its importance. We have also done a comparison of the VC-hierarchy with other related hierarchies in parameterized complexity domain.

3.1 Definitions and other important notions

VC-hierarchy was first defined by Harnik and Naor ([13]). But we have changed it a little bit to fit the definitions for parametric problems with natural witness length as the parameter. In the original definitions, NP languages are considered. But that does not fit directly to the parametric problems that we have considered so far. So here we have extended those definitions for parametric problems in NP where the parameter can be interpreted as the *witness size* for some natural NTM deciding the language (as we did in previous chapter).

Before defining some of the problems in this context, we would like to mention that by a circuit of depth k we mean a circuit which consists of k alternating *AND* and *OR* gates where the fan-in of the gates are bounded only by the circuit size and the *NOT* gates are present only on the input variables. Let us now consider the following definition.

Definition 25. DEPTH$_k$CIRCUITSAT *(where $k \geq 2$):*
***Input**: A circuit C of size m and depth at most k over n variables.*
***Membership**: $C \in$ DEPTH$_k$CIRCUITSAT if there exists a satisfying assignment to C.*
***Parameter**: n*

Definition 26. LOCALCIRCUITSAT*:*
***Input**: A string x of length m and a circuit C over (n + n.log m) variables and of size O(n + n.log m).*
***Membership**: (x, C) $\in$ LOCALCIRCUITSAT if there exists a list I of n locations in x such that $C(\langle x(I), I\rangle) = 1$.*
***Parameter**: n + n.log m*

Definition 27. ***The VC classification of parametric problems**: Consider parametric problems in NP where m denotes the instance size and n denotes the parameter. We define the class VC_k for every $k \geq 0$. The definition is divided into three cases:*

- *$k = 0$: The class VC_0 is the class of all parametric problems that admit compression algorithms.*
- *$k = 1$: The class VC_1 is the class of all parametric problems that W-reduce to* LOCALCIRCUITSAT.
- *$k \geq 2$: The class VC_k is the class of all parametric problems that W-reduce to* DEPTH$_k$CIRCUITSAT.

For any function $k(m,n)$ (where $k(m,n) \leq m$) also define $VC_{k(m,n)}$ as the class of all parametric problems that W-reduce to $Depth_{k(m,n)}CircuitSAT$. Finally, define $VC = VC_m$ (the class for $k(m,n) = m$).

A parametric problem L is called *compression-hard* for a class VC_k ($k \geq 2$), if there is a w-reduction from DEPTH$_k$CIRCUITSAT to L. If L is also present in VC_k, it is called *compression-complete* for VC_k. Analogously, we can define the compression-hardness and completeness for the classes VC_1 and VC_{OR}.

Definition 28. ***Verification with Preprocessing****: Let L (* $\subseteq \{\langle x, 1^n\rangle | x \in \{0,1\}^*, n \in \mathbb{N}\}$ *) be a parametric problem with instance size* $|x| = m$ *and parameter n. A pair of polynomial-time algorithms* (P,V) *is called a verification with preprocessing for L if the following two-step verification holds:*

- ***Preprocessing****: P gets an instance* $\langle x, 1^n\rangle$ *and outputs a new instance* $P(\langle x, 1^n\rangle)$.
- ***Verification****: There exists a polynomial* $p(.)$ *such that* $\langle x, 1^n\rangle \in L$ *if and only if there exists a witness w of length at most* $p(n)$ *such that* $V(P(\langle x, 1^n\rangle), w) = 1$.

We can observe that when we are allowing preprocessing, all problems solvable in polynomial time have a pair (P,V) where P solves the problem and stores the answer while V simply accepts this answer to be correct. So if we consider the complexity of V in this definition, we can see that the easy problems indeed have very low complexity.

The *VC* Classification via *Verification* with *Preprocessing*: An alternative and equivalent way to view the classes in the VC hierarchy is based on the *verification* algorithm V in a *verification* with *preprocessing* pair (P,V) [13]. The problems are divided into two families:

- The class VC_1 is the set of the parametric problems that have very efficient verification (i.e., $poly(n)$ rather than $poly(m)$). We assume random access to the instance thus such a verification algorithm only accesses a sub-linear fraction of the instance.
- The parametric problems whose verification is not very efficient (run in time $poly(m)$). This family is further classified into sub categories. The class VC_k is the class of parametric problems where the verification algorithm V has a representation as a depth k *polynomial* size circuit (polynomial in m).

This second definition is equivalent to the definition via w-reductions in the following way. The w-reduction to the complete problem can simply be viewed as the preprocessing stage. In the other direction, every preprocessing stage is actually a w-reduction to the parametric problem defined by V.

We are now going to re-consider the OR-L problem (section 2.1) more formally.

Definition 29. OR-L*: Let L be a parametric problem in NP. We define the parametric problem* OR-L *as follows:*

Input: *m instances* $x_1, \ldots, x_m$ *of the problem L, each of length n.*
Membership: ($x_1, \ldots, x_m$) $\in$ OR-L *if there exists* $i \in [m]$ *such that* $x_i \in L$.
Parameter: $n + log(m)$

The class VC_{or} is the class of all parametric problems that w-reducible to the problem OR-CIRCUITSAT.

The following relation is already shown in [13].

Theorem 8. $VC_0 \subseteq VC_{or} \subseteq VC_1 \subseteq VC_2 \subseteq VC_3 \subseteq \ldots$.

3.2 VC hierarchy classification of some natural problems

In this section we are going to place some natural parametric problems in the VC hierarchy. We will also present some hardness and completeness results. Some of the problems are already classified in [13]. We will show some improvements of them as well. We start with some of the existing results. We are going to see the formal definitions of these problems gradually. (All problems used here are defined together in the appendix, section 3.5.)

* CLIQUE is *Compression-hard* for VC_{or} [13]

* SAT is *Compression-complete* for VC_2 [13]

We are now going to classify some of the problems with respect to their difficulty in compression.

3.2.1 Positions of some of the parametric problems

We start with the definitions of some simple problems. They are not simple in terms of solvability. But comparatively simpler in terms of compression as they are present in the lower levels of VC hierarchy.

k-COLOURABILITY PROBLEM:

Input: A graph $G(V,E)$ with n vertices, and an integer k.
Task: Decide whether G is k colourable.
Parameter: $n.log(k)$

k-CYCLE:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a cycle of length k.
Parameter: $k.log(n)$

k-COLOURABILITY PROBLEM $\in VC_0$: In this problem we try to find the assignments of k colours in n vertices of graph $G(V,E)$ ($|V| = n$), such that no two adjacent vertices have same colours. More formally, we try to find a function $C : V \rightarrow \{1,2,\ldots,k\}$ such that, $(u,v) \in V$ and $C(u) \neq C(v)$. We can encode any such colouring (natural witness of the problem) in $n.log(k)$ bits. So here the parameter of the problem k-COLOURABILITY PROBLEM is $n.log(k)$. Instance size $|G|$ is clearly bounded polynomially by the parameter $n.log(k)$. So by definition G is already compressed. So this NP parametric problem belongs to VC_0. □

k-CYCLE $\in VC_1$: In k-CYCLE problem we try to find a proper ordering of k vertices in a graph such that they construct a cycle. In a graph $G(V,E)$, $|G| = m$ is the instance size (which is polynomially bounded in both $|V|$ and $|E|$). A k-cycle consists of k vertices in order. So any witness can be encoded in $k.log(n)$ bits. Hence, the parameter $K = k.log(n)$. Now, whether these k vertices are forming k-cycle or not, can be easily verified in $poly(K)$ time. We don't access the whole graph. So this NP parametric problem belongs to VC_1. □

We now consider slightly more interesting problems.

SUB-GRAPH ISOMORPHISM:

Input: Two graphs $G = (V_1, E_1)$ with n vertices, and $H = (V_2, E_2)$ with k vertices.
Task: Decide whether G contains a sub-graph isomorphic to H (a subset $V \subseteq V_1$ and a subset $E \subseteq E_1$ such that $|V| = |V_2|$, $|E| = |E_2|$, and there exist a one-to-one function f: $|V_2| \rightarrow |V|$ satisfying $\{u,v\} \in E_2$ iff $\{f(u), f(v)\} \in E$).
Parameter: $k.log(n)$

CLIQUE:

Input: A graph G with n vertices, and an integer k.
Task: Decide whether G has a clique of size k (a pairwise adjacent subset of k vertices).
Parameter: $k.log(n)$

SUB-GRAPH ISOMORPHISM $\in VC_1$ and w-reducible from CLIQUE: The sub-graph isomorphism problem is a decision problem in which two graphs G and H are given as input, and one must determine whether G contains a sub-graph that is isomorphic to H. This problem can be polynomially reduced from CLIQUE problem and eventually this reduction is w-reduction. In CLIQUE problem the input is a single graph

G and a number k, and the question is whether G contains a complete sub-graph with k vertices or not. To translate this to a SUB-GRAPH ISOMORPHISM problem, we will simply take H to be the complete graph with k vertices. Then the answer to the sub-graph isomorphism problem for G and H is equal to the answer to the clique problem for G and k. Here the witness for the sub-graph isomorphism is actually the mapping of k vertices of H in G. Clearly, the witness size is polynomially bounded in $k.log(n)$. So CLIQUE problem is w-reducible to SUB-GRAPH ISOMORPHISM. As we already know that CLIQUE is *compression-hard* for VC_{or} [13], then SUB-GRAPH ISOMORPHISM is also *compression-hard* for VC_{or}.

Proof of the containment in VC_1 is rather easy. Once the mapping of k vertices of H in G is given, the verification can be done easily in $poly(k.log(n))$ time. We do not need to consider the whole graph G for verification. So clearly this NP problem belongs to VC_1. □

We now consider a few similar problems.

LARGEST COMMON SUBGRAPH:

Input: Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ and an integer k. Suppose n is the maximum of the numbers of vertices in these two graphs.

Task: Decide whether there exist subsets $V_1^{'} \subseteq V_1$ and $V_2^{'} \subseteq V_2$ with $|V_1^{'}| = |V_2^{'}| = k$ such that two vertex induced subgraphs $G^{'} = (V_1^{'}, E_1^{'})$ and $H^{'} = (V_2^{'}, E_2^{'})$ are isomorphic.

Parameter: $k.log(n)$

MAXIMUM SUBGRAPH MATCHING:

Input: Two directed graphs $G = (V_1, A_1)$ and $H = (V_2, A_2)$ and an integer k. Suppose n is the maximum of the numbers of vertices in these two graphs.

Task: Decide whether there is a subset $R \subseteq V_1 \times V_2$ with $|R| = k$ such that for all $\langle u, u^{'} \rangle$, $\langle v, v^{'} \rangle \in R$, $(u, v) \in A_1$ iff $(u^{'}, v^{'}) \in A_2$.

Parameter: $k.log(n)$

In the similar way we can prove the w-reductions from SUB-GRAPH ISOMORPHISM to LARGEST COMMON SUBGRAPH and MAXIMUM SUBGRAPH MATCHING and their containment in VC_1. In fact, we can also conclude that these two new problems are *compression-hard* for VC_{or}. Where the reduction to LARGEST COMMON SUBGRAPH is very similar, for the reduction to MAXIMUM SUBGRAPH MATCHING, we will simply replace every undirected edge of both the graphs by 2 bi-directional edges.

Let us now consider another similar type of problem.

BALANCED COMPLETE BIPARTITE SUBGRAPH:

Input: A Bipartite graphs $G = (V, E)$ with n vertices and an positive integer $k \leq |V|$.

Task: Decide whether there are two disjoint independent sets $V_1, V_2 \subseteq V$ such that $|V_1| = |V_2| = k$ and $u \in V_1$, $v \in V_2$ implies that $\{u, v\} \in E$.

Parameter: $k.log(n)$

We can see that w-reduction from BALANCED COMPLETE BIPARTITE SUBGRAPH to SUB-GRAPH ISOMORPHISM, LARGEST COMMON SUBGRAPH, and MAXIMUM SUBGRAPH MATCHING are also similar. Hence the BALANCED COMPLETE BIPARTITE SUBGRAPH problem is in VC_1.

We can also consider simple *Graph Isomorphism* problem.

GRAPH ISOMORPHISM:

Input: Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ both with n vertices.

Task: Decide whether G and H are isomorphic to each other.

Parameter: $n.log(n)$

It is easy to see that GRAPH ISOMORPHISM problem is in VC_0, as in that case *instance size* is polynomially bounded with respect to the *parameter* of the problem.

We now consider slightly different type of problems.

SUBSET SUM:

Input: A set S of m non-negative integers, and another integer B.

Task: Decide whether there is a subset $S^{'} \subseteq S$ such that the summation of all the elements is $S^{'}$ is B.

Parameter: $|S^{'}|.log(m)$

For solving SUBSET SUM problem, we are given a set of m values and a target sum B. In this problem, the objective to find a subset $S^{'}$ (total size $O(|S^{'}|.log(m))$) from the set S such that the values from the subset add up exactly to the target sum B. So the parameter K is $O(k.log(m))$ if $k = |S^{'}|$. Suppose we are given such a witness of k values, the verification may be done easily in $poly(K)$ time rather than in $poly(m)$ time, just by adding those k values. We may not need to consider the whole set of size m for verification. But it does not mean that we can put this NP problem into VC_1. Because, the above mentioned verification can be done in $poly(K)$ only if we assume that the size of each of the m integers are constant. But that is not the general case. Hence we can not place the problem into VC_1. In fact, we can not add k integers by a circuit of constant depth when the size of the integers are not constant. Hence, we can not place them in VC_t for any constant t. Nevertheless, variations of this SUBSET

SUM problem are quite interesting and we are going to consider this problem in much more details later.

We now consider following two popular problems.

INDEPENDENT SET:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has an independent set of size k (a pairwise non-adjacent subset of k vertices).

Parameter: $k.log(n)$

SET PACKING:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set) and an integer k.

Task: Decide whether there is a set packing of size k (a subset $S \subseteq E$ such that, for all distinct pair of sets $S_1, S_2 \in S$, S_1 and S_2 are mutually exclusive).

Parameter: $k.log(m)$

INDEPENDENT SET problem is w-reducible to SET PACKING: In SET PACKING problem, we have a finite set V and a list of subsets E (= $\{S_1,S_2, \ldots, S_m\}$) of S. Then, the SET PACKING problem asks if there is some subset of size k from the list such that the elements of the subset are pairwise disjoint (in other words, no two of them have non-null intersection). There is a very simple w-reduction from INDEPENDENT SET to SET PACKING problem. Suppose a graph G is given to us which is our INDEPENDENT SET instance. We produce the sets S_1,S_2, …, S_m as follows: we generate a set S_i for each vertex v_i in G, so that the set contains exactly the edges e incident on v_i. Clearly, m = number of vertices of G. V of the SET PACKING instance is just the set of all edges in G. Finally, for this SET PACKING problem, we ask if there is a set packing of size k or not.

Now it is not difficult to see that the INDEPENDENT SET instance G has an independent set of size k iff SET PACKING problem instance (V,E) has a set packing of size k. For proving this claim, suppose G has an independent set of size k. Let us name this set as IS. We now construct S by including exactly the S_i's corresponding to $v_i \in IS$. Obviously, the size of S is equal to the k. Since no two vertices in S can share an edge, the sets S_i we picked must be pairwise disjoint, and hence we have found a valid set packing of k sets.

Now, assume that we have a set packing S of size k for the SET PACKING problem instance (V,E). Then, we construct a vertex set IS by taking exactly those v_is for which

$S_i \in S$. The size of IS is the same as that of S. For any edge e, there is at most one set S_i with $e \in S_i$ and $S_i \in S$. So at most one node $v_i \in IS$ can be incident on any edge e. Thus, no two selected nodes are connected by an edge, and hence IS is in fact an independent set of size k.

So INDEPENDENT SET problem is w-reducible to SET PACKING. As we already know that INDEPENDENT SET is *compression-hard* for VC_{or} (as INDEPENDENT SET and CLIQUE are equivalent with respect to common w-reduction), then SET PACKING is also *compression - hard* for VC_{or}. □

There are a few interesting problems which have polynomial size kernels. This shows that those problems are compressible and hence in VC_0. For example, FEEDBACK ARC SET on tournament has a linear size Kernel [51]. FEEDBACK VERTEX SET has a cubic Kernel on undirected graph [19]. As mentioned in chapter 1, *polynomial kernelization* (where the *problem kernel* size is a polynomial function of the initial problem parameter) is equivalent to the deterministic compression to size $poly(n)$ (n is the parameter of the initial problem instance). In case of compression, the parameter is chosen in such a way that it can be interpreted as the *witness size* for some natural NTM deciding the language. We have not considered the above mentioned VC_0 problems here and hence are not going to discuss them in details. We are more interested in those problems which are not known to be compressible. That is why, we are now going to consider some higher levels of VC-hierarchy and corresponding natural problems in more details.

3.2.2 VC_1 and corresponding problems

We are now going to consider the class VC_1 in more details. This is the class of parametric problems which are locally verifiable. That means, we can verify the witness of such problem instance in time polynomial in parameter where we take some natural witness length as parameter. We can understand that a large number of popular natural problems belong to this class, which makes this class very interesting. We have already considered a few of them before.

We are now going to consider a few more problems here which are related to this interesting class VC_1. We first consider the primary completeness results for VC_1 and then finally, we will summarize the positions of several natural parametric problems with respect to VC_1. In many cases, we have considered different variations of the same

problem. The problem with seemingly more natural parameter are named without any subscript. Other variations are named with some subscript depending on the definition and parameter of the problem. We have followed same convention throughout this work (especially in this chapter and chapter 4).

We are going to prove the first completeness result for VC_1 using the idea used in [14], though the parameter choice was different in their work. We first define a new problem and prove the next result.

BINARY NDTM HALTING:

Input: The code of a Turing machine M of size n with a binary alphabet, and an integer k.

Task: Decide whether M halts on the empty string in k steps.

Parameter: $k.log(n)$

In the original work ([14]), parameter was k instead of $k.log(n)$.

Proposition 6. BINARY NDTM HALTING *is* VC_1*-complete.*

Proof. Firstly we are going to prove the membership of the BINARY NDTM HALTING problem in VC_1. In this problem, a Turing machine M of size n with a binary alphabet, and an integer k are given. We want to decide whether M halts on the empty string in k steps. Parameter K is $k.log(n)$. If k transitions are given as the witness, we can verify if that given k steps can lead the machine to halting state. So clearly the verification can be done in $poly(K)$ time. Hence, the membership in VC_1 is proved.

To prove the hardness result, we are going to show a w-reduction from LOCAL-CIRCUITSAT to this problem (as shown in [14], Theorem 5). In LOCALCIRCUITSAT a string x of length m and a circuit C over ($t + t.log\ m$) variables and of size $O($ $t + t.log\ m$) are given. We decide whether there exists a list I of t locations in x such that $C(x(I), I) = 1$. Parameter is $t + t.log(m)$. Without loss of generality we can assume that the circuit size is polynomially bounded by $t.log(m)$. To show the reduction we are going to use the well known result which says that a Turing machine can be constructed to simulate a fixed circuit in a number of steps that is polynomial in the circuit size ([23], chapter 3 and [5], chapter 9). So clearly we can take $k = poly(t.log(m))$. To reduce a LOCALCIRCUITSAT instance, in the beginning we will encode the input string of length m into the Turing machine. So we can easily see that the size of the Turing machine n will be polynomially bounded by m. Now the machine nondeterministically chooses the t positions in $k = poly(t.log(m))$ steps and writes the

corresponding contents of the string onto the tape, followed by the positions. After that the circuit will be simulated by the Turing machine according to the well known result mentioned above. Here the parameter of the BINARY NDTM HALTING problem K will be $k.log(n) = poly(t.log(m).log(m)) = poly(t.(log(m))^2)$. Clearly it is a w-reduction. Hence BINARY NDTM HALTING is VC_1-complete. □

We are now going to present our next completeness result. In [13], it was pointed out that many natural problems are in VC_1 and asked if any one of them can be proved as VC_1-complete. Our next result answers that question using some of the techniques used in [14].

Theorem 9. CLIQUE *is* VC_1*-complete.*

Proof. It is already known that CLIQUE is present in VC_1 ([13]). Now to show the hardness result, we are going to show a w-reduction from BINARY NDTM HALTING to CLIQUE problem, as BINARY NDTM HALTING is already proved to be VC_1-complete in the Proposition 6. Now, we are going to show the reduction in three steps. Firstly we will show a w-reduction from BINARY NDTM HALTING to WEIGHTED 3-CNF SATISFIABILITY. Then w-reduction from WEIGHTED 3-CNF SATISFIABILITY to ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY. And in the end w-reduction from ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY to CLIQUE. Let us first see the definitions of the newly introduced problems.

WEIGHTED 3-CNF SATISFIABILITY

Input: A formula ϕ in 3-*CNF* (each clause contains at most 3 literals) with n input variables, and an integer k.

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY

Input: A formula ϕ in 2-*CNF* (each clause contains at most 2 literals) with n input variables where all the literals in ϕ are negative literals, and an integer k.

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

Formal definitions of the problems are given above. Now for proving the reduction in the first step (w-reduction from BINARY NDTM HALTING to WEIGHTED 3-CNF SATISFIABILITY), we are going to use the reduction used by Hermelin et. el.(Lemma

13, [14]). Though the parameter used there for BINARY NDTM HALTING is different from our parameter, it can be checked that the same reduction will work here and the reduction will be still a *w*-reduction with our parameter. Although this technique has similarity with the proof of the well known Cook's theorem ([52]), this reduction is *w*-reduction where the Cook's reduction is not. We are now briefly going to discuss that reduction here to understand that the above mentioned claim is correct.

Suppose the code of a Turing machine M of size n with a binary alphabet and an integer k are given. We also assume that M has s states with l edges in its transition diagram. We now construct a 3-*CNF* formula ϕ such that ϕ has a satisfying assignment of weight $k^{'}$, iff M accepts the empty string in k steps. We will choose $k^{'}$ as a polynomial function of k as mentioned later. To construct the formula ϕ, we will use following set of variables.

- Variables $S_{i,t}$, $i \in [s]$ and $t \in \{0\} \cup [k]$, signifying that the machine is in state number i after t time steps.
- Variables $M_{e,t}$, $e \in [l]$ and $t \in [k]$, signifying that edge e of the machine's state diagram is followed as step t.
- Variables $H_{p,t}$, $0 \leq p, t \leq k$, signifying that the machine's tape head is in position p after t time steps.
- Variables $T_{p,t}$, $0 \leq p, t \leq k$, signifying that position p of the tape contains value 1 after t time steps.
- Variables $\overline{T_{p,t}}$, $0 \leq p$, $t \leq k$. These are constrained so that $T_{p,t} \neq \overline{T_{p,t}}$ for all values of p and t, allowing us to control the weight of any satisfying assignment for ϕ.

We will now add clauses into the formula ϕ to maintain all the constrains of the Turing machine. We firstly add 2 clauses of size 2 as follows, $(T_{p,t} \vee \overline{T_{p,t}})$ and $(\neg T_{p,t} \vee \neg\overline{T_{p,t}})$, for all values of p and t. It is done to ensure that $T_{p,t} \neq \overline{T_{p,t}}$ for all values of p and t. Then we will ensure that at most one variable among $S_{.,t}$, $H_{.,t}$, and $M_{.,t}$ is *True* for each t. We do that by creating a conjunction of clauses of size 2 over the corresponding variable sets (for each pair of variables, we take disjunction of negations of them to ensure two of them can not be *True* together). We additionally enforce that the final state of the machine is an accepting state (by adding clauses of size 1, containing only one negative literal). We then add clauses enforcing consistency of

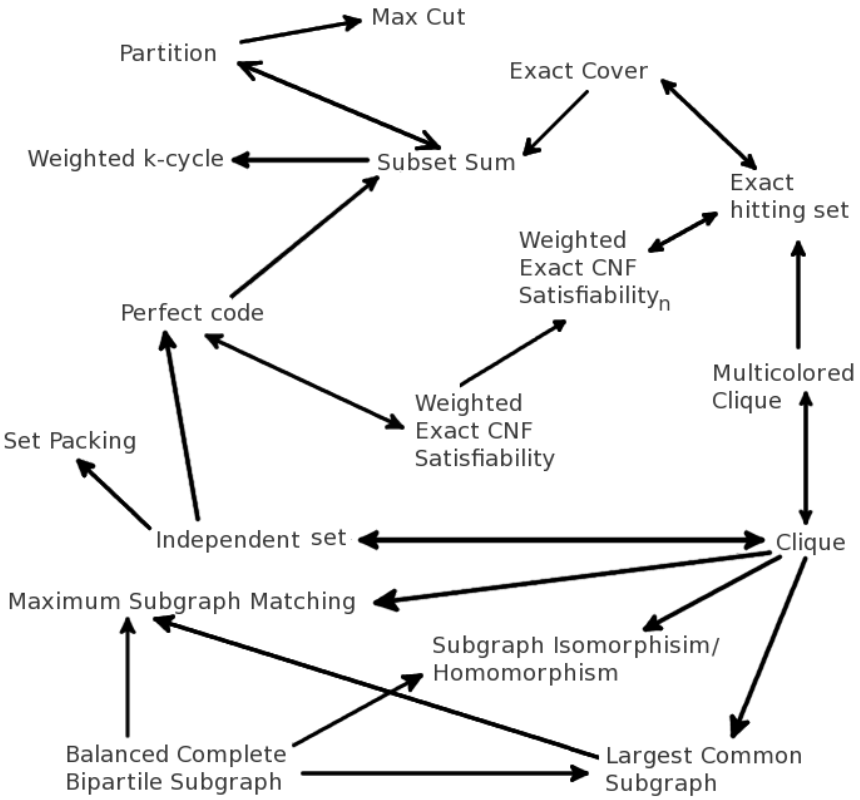
states, head, tape, and transition variables as shown by Hermelin et. el.(Lemma 13, [14]). Finally, we will add clauses to encode $\neg H_{p,t} \rightarrow (T_{p,t} = T_{p,t+1})$ for all p and t and clauses to control the initial set up of the machine. We can see that the number of literals in any clause of the formula ϕ is at most 3. As a result, number of variables and the size of the formula ϕ, both are polynomially bounded in n. It is not very difficult to see that we can choose our above mentioned $k^{'}$ as $2(k+1)+(k+1)^2+k$ and hence the reduction is actually *w*-reduction. More detailed proof for this reduction can be found in [14] (Lemma 13).

To show the second step (*w*-reduction from WEIGHTED 3-CNF SATISFIABILITY to ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY), again we are going to use the reduction used by Hermelin et. el.(Lemma 2, [14]). There, from a *t-CNF* formula ϕ (each clause contains at most t literals) for any constant t, they have constructed another formula ψ, where ψ is in 2-*CNF* with all negative literals (using the tricks used in [25]). It can be observed that ϕ has a satisfying assignment of weight k iff ψ has a satisfying assignment of weight $k^{'}$, where $k^{'}$ is polynomially bounded in k. The parameter of the reduced instance is polynomially bounded with the initial one and hence, it is a *w*-reduction. Taking t = 3, *w*-reduction from WEIGHTED 3-CNF SATISFIABILITY to ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY directly follows from that result (Lemma 2, [14]).

Now to prove the last step, we are going to use a very simple reduction. From the given ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY instance (a formula ϕ in 2-*CNF* with n input variables), we will construct a graph. We will take a vertex in the graph G for each variable in the formula ϕ, the ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY instance. Then we will add an edge in the graph G corresponding to every clause of size 2. Single literal clauses can be easily removed in polynomial time before constructing this graph, by assigning required values to the variables. In this process, if the problem instance is solved to be a *NO* instance, we can out put a trivial graph with no independent set of size k. Otherwise, it is easy to see that ϕ is satisfiable with weight k iff the graph G has an independent set of size k. INDEPENDENT SET is well known to be *w*-equivalent to CLIQUE. Hence CLIQUE is VC_1-complete. □

We are now going to consider a series of popular NP problems and their relations with respect to *w*-reduction. They will help us to prove some new hardness or completeness results. Before going into formal definitions of the problems and proof for the reductions, we will give a brief summary about what we going to see next. The

Figure 3.1: w-reductions among parametric problems



problems we have considered here are present in Figure 3.1. In this diagram, arrow from the problem A to B implies that we claim that there is a w-reduction from problem A to B. We are going to show the reductions soon.

We also see from the diagram (Figure 3.1) that, though most of the problems are quite natural and popular, MULTICOLOURED CLIQUE is not that natural. But it is an important problem as it is connecting CLIQUE with many other natural problems such as, EXACT HITTING SET, EXACT COVER etc. more directly. For the same reason we have considered WEIGHTED EXACT CNF SATISFIABILITY problem with two different parameters. They are really the key problems to prove some of the completeness results as we will see soon. To prove the reductions, in many cases we use the standard reductions and either verify or modify them slightly to see that they are essentially w-reductions. But in some other cases we have to work a bit more, e.g. reduction from WEIGHTED EXACT CNF SATISFIABILITY$_n$ to EXACT HITTING SET. There are several interesting techniques we will see in these reductions. Some of them use graph theory tricks (reductions related to CLIQUE, MAX CUT, PERFECT CODE etc.), some others use Boolean formula manipulation (reductions connecting WEIGHTED EXACT

CNF SATISFIABILITY) or in some cases simple but useful set theory or number theory techniques (connections among the problems EXACT HITTING SET, EXACT COVER, SUBSET SUM etc.). We are now going to discuss them in more details.

We have already considered the problem SUBSET SUM. Let us now define another similar problem PARTITION.

PARTITION:

Input: A set S of m non-negative integers.

Task: Decide whether there is a set $S^{'} \subseteq S$ such that, $\sum_{a \in S'} a = \sum_{a \in S-S'} a$.

Parameter: $(min(|S^{'}|, |S - S^{'}|)).log(m)$

We have considered SUBSET SUM before and explained why we can not put it into VC_1. We are going to consider the same problem once again to prove the following simple result.

Proposition 7. SUBSET SUM *and* PARTITION *problems are w-reducible to each other.*

Proof. In SUBSET SUM problem, a finite set S of non-negative integers are given with a target integer B. For this problem instance, we ask if there exists a subset $S^{'} \subseteq S$ such that, $\sum_{a \in S'} a = B$. Here S contains m non-negative integers. The elements in $S^{'}$ are the witness. If $k = |S^{'}|$, it is easy to understand that the parameter $K = k.log(m)$ as $k.log(m)$ bits can capture a natural witness.

Similarly for PARTITION problem it is easy to see that, if $k = min(|S^{'}|, |S - S^{'}|)$, the parameter K of the problem can be $k.log(m)$ as $k.log(m)$ bits can capture a natural witness.

w-reduction from PARTITION *to* SUBSET SUM*:* This reduction is very much straight forward. We can take $B = (\sum_{a \in S} a)/2$, from the PARTITION problem for SUBSET SUM. Clearly we can calculate B in $poly(m)$ and the reduction is a w-reduction.

w-reduction from SUBSET SUM *to* PARTITION*:* To show this reduction (original reduction shown in [45]), we introduce two new variables of values $T - B$ and $T - (\sum_{a \in S} a - B)$ in our set. Let us consider the new set is named as S^*. We will choose T to be sufficiently large. Any $T > \sum_{a \in S} a$ will work for this reduction. But at the same time, T should be sufficiently small so that it is a w-reduction. We will choose T to be just large enough so that both the new elements can not go to the same partition as in that case they will overweight all the other elements. Now we ask whether this new set S^* can be partitioned in two sets of equal summation of values within it. Clearly the individual summation will be of values T as $\sum_{a \in S^*} a = 2T$. If there exists a partition for the set S^*, after taking out the new two values from both the partitions, clearly we will

get a partition of original set S into two sets of total summation B and $(\sum_{a \in S'} a - B)$. Hence, it is now easy to see that, there exists a PARTITION for S^* iff there exists a SUBSET SUM of S for target value B. Clearly this reduction is w-reduction as we are introducing a constant number (which is 2 here) of new values where the size of the values are polynomially bounded in problem size. □

We are now going to consider another popular problem.

EXACT COVER:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set).

Task: Decide whether there is an exact cover $S \subseteq E$ for V (Exact cover is a subset $S \subseteq E$ of pairwise disjoint sets with $\bigcup S = V$).

Parameter: $|S|.log(m)$

We are now going to prove the following.

Proposition 8. EXACT COVER *is w-reducible to* SUBSET SUM.

Proof. We have used the standard polynomial reduction from EXACT COVER to SUBSET SUM ([45]) to show this result.

Let us consider that $V = \{t_0, t_1, \ldots, t_{n-1}\}$ in the given instance of *Exact Cover (V, E)*. Number of elements in E (i.e., the collection of the subsets of V) is m. Now for $t_x \in V$, $0 \leq x \leq n-1$, let us define,

$\#x = |\{A \in E | t_x \in A\}|$, the number of elements of E containing t_x, $0 \leq x \leq n-1$.

Let us consider p to be a number just exceeding all $\#x$, $0 \leq x \leq n-1$. We will keep the size of p polynomially bounded in problem size. Next, we encode $A \in E$ as follows,

$w(A) = \sum_{t_x \in A} p^x$, for all $t_x \in V$.

and take

$B = \sum_{x=0}^{n-1} p^x = (p^n - 1)/(p-1)$.

We consider the set $S = \{w(A) | A \in E\}$. In p-ary notation $w(A)$ looks like a string of 0 and 1 with 1 in position x ($0 \leq x \leq n-1$) for each $t_x \in A$ and 0 elsewhere. The number B in p-ary notation looks like a string of length n containing all 1. Adding the numbers $w(A)$ simulates the union of the sets A. The number p was chosen big enough so that we do not get into trouble with carries. Now asking whether there is a SUBSET SUM of the set S that gives B, is the same as asking for an EXACT COVER for V. Clearly the same sets within the collection E for the construction of an EXACT COVER will correspond to the subset $S' \subseteq S$ to get the sum B. Parameter for the SUBSET SUM

problem instance is $|S^{'}|.log(m)$. So clearly this is a w-reduction from EXACT COVER to SUBSET SUM. □

We are now going to consider another interesting problem. It is interesting because of the parameter choice.

SET COVER$_{kn}$:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set) and an integer k.

Task: Decide whether there is a set cover of size k (a subset $S \subseteq E$ with $\bigcup S = V$).

Parameter: kn

Here kn is a valid parameter as one can encode any set in E in n bits.

We now prove the following.

Proposition 9. SET COVER$_{kn}$ *is in* VC_1 *and w-reducible to* SUBSET SUM.

Proof. It is not difficult to see that SET COVER$_{kn}$ is in VC_1. Any witness consisting of k sets, each encoded in n bits, can be easily checked in $poly(kn)$ time to verify if they are forming a set cover or not. Hence, SET COVER$_{kn} \in VC_1$.

We will prove this result by showing a reduction which is quite similar to the result shown just before it. Let's consider that $V = \{t_0, t_1, \ldots, t_{n-1}\}$ in the given SET COVER$_{kn}$ instance (V, E). For this problem instance, we ask if there is a set cover of size k. Parameter for this problem is kn. Number of elements in E (i.e., the collection of the subsets of V) is m. Now for $t_x \in V$, $0 \leq x \leq n-1$, let us define,
$\#x = |\{A \in E | t_x \in A\}|$, the number of elements of E containing t_x, $0 \leq x \leq n-1$.
Let us consider p to be a number just exceeding all $\#x$, $0 \leq x \leq n-1$ and k. We will keep the size of p polynomially bounded in problem size. For each $A \in E$ we define,
$w(A) = \sum_{t_x \in A} p^x + p^n$, for all $t_x \in X$.

In p-ary notation $w(A)$ looks like a string of 0 and 1 with 1 in position x ($0 \leq x \leq n-1$) for each $t_x \in A$ and in position n with 0 elsewhere. Here the n^{th} bit will ensure that the size of the set cover is always k.

We also introduce following extra elements as follows.
$w(B_{i,j}) = p^i$, for all $0 \leq i \leq n-1$, $1 \leq j \leq k-1$.

It means, we are introducing extra $(k-1)n$ elements. We can also see that $w(B_{i,j})$ looks like a p-ary number of length $n+1$ where values are zeroes at all the positions except at i^{th} ($0 \leq i \leq n-1$) position where it is 1. We can also consider that $B_{i,j}$ is a set containing just one element t_i and $w(B_{i,j})$ is the decimal value for its p-ary representation.

We now take
$B = \sum_{x=0}^{n} kp^x = k(p^{n+1}-1)/(p-1)$.
We consider the set $S = \{w(A)|A \in E\} \cup \{w(B_{i,j})|0 \leq i \leq n-1, 1 \leq j \leq k-1\}$. The number B in p-ary notation looks like a string of length $n+1$ containing all k. Adding the numbers in S simulates the union of the sets $A \in V$ and $B_{i,j}$, $0 \leq i \leq n-1, 1 \leq j \leq k-1$. The number p was chosen big enough so that we do not get into trouble with carries. Now asking for a *Set Cover* for V of size k, is the same as asking whether there is a *Subset Sum* (of size at most $k^{'} = k + (k-1)n$) of the set S that gives B. Clearly the same sets within the collection E for the construction of a *Set Cover* of size k will correspond to the subset to get the sum B with some extra elements from the set $\{w(B_{i,j})|0 \leq i \leq l-1, 1 \leq j \leq k-1\}$. Parameter for the SUBSET SUM problem instance is $k^{'}.log(|S|)$. So clearly this is a w-reduction from SET COVER$_{kn}$ to SUBSET SUM. □

We now define the following.

MAX CUT:

Input: A weighted graph $G(V,E)$ with m edges and n vertices, an integer B.
Task: Decide whether there exists a subset $V^{'} \subseteq V$ such that the total size of the cut (sum of weights of edges between vertex sets $V^{'}$ and $V - V^{'}$) is $\geq B$.
Parameter: $|V^{'}|.log(n)$

We can not place this problem into VC_1 as to check if the cut size is equal to B or not, we have to consider the cross edges between $V - V^{'}$ and $V^{'}$. Hence, the algorithm run time will not be $poly(K)$ where K is the parameter of the problem instance. But we can prove the following. This particular reduction was first pointed out in [45].

Proposition 10. PARTITION *is w-reducible to* MAX CUT.

Proof. Suppose we have a PARTITION instance, a set S of m non-negative integers. From this S, we are going to construct a graph $G(V,E)$ as follows. For each integer $a_i \in S$, $1 \leq i \leq m$, we will construct a vertex v_i of the graph G. We connect every pair of distinct vertices $(u_i, v_i), i \neq j$, from V by an edge with weight $a_i a_j$. We take the cut size for the MAX CUT problem as $B = (\sum_{i=1}^{m} a_i)^2/4$.

Firstly, we can see that, any subset of S, map a subset of vertices from the vertex set V of G. If there is a partition $S^{'}$ for S, such that $\sum_{a \in S^{'}} a = \sum_{a \in S-S^{'}} a$, $S^{'}$ it will similarly map a subset $V^{'} \subseteq V$. Moreover, it is now easy to see that, any such partition $S^{'}$ will result in a cut between $V - V^{'}$ and $V^{'}$ of size $\sum_{a \in S^{'}} a \times \sum_{a \in S-S^{'}} a = ((\sum_{i=1}^{m} a_i)/2) \times ((\sum_{i=1}^{m} a_i)/2) = (\sum_{i=1}^{m} a_i)^2/4$. So in one direction proof is done.

Now suppose, there is subset $V^{'} \subseteq V$ such that the total size of the Cut with respect to the rest of the vertices $V - V^{'}$ is $B = (\sum_{i=1}^{m} a_i)^2/4$. Similarly, any such subset $V^{'} \subseteq V$ will map a subset $S^{'} \subseteq S$. We name this mapping as f. Now we claim that, the maximum size of any cut in graph G is $(\sum_{i=1}^{m} a_i)^2/4$ and it happens only when the vertices of V is partitioned into $V^{'}$ and $V - V^{'}$ in such a way that $\sum_{a \in f(V^{'})} a = \sum_{a \in f(V - V^{'})} a$. We prove this claim as follows.

Suppose V is partitioned into $V^{'}$ and $V - V^{'}$. It is now easy to see that the total size of cut in this case is $\sum_{a \in f(V^{'})} a \times \sum_{a \in f(V - V^{'})} a$. We take $f(V^{'}) = S^{'}$, $f(V - V^{'}) = S - S^{'}$ as $f(V) = S$ and also $\sum_{i=1}^{m} a_i = T$. So we have to partition the set S, i.e., total value T in such a way that cut size $C = \sum_{a \in S^{'}} a \times \sum_{a \in S - S^{'}} a = \sum_{a \in S^{'}} a \times (T - \sum_{a \in S^{'}} a)$ is maximum ($S^{'}$ and $S - S^{'}$ are the partition of S). We take $\sum_{a \in S^{'}} a = x$. Hence, $C = x(T - x) = Tx - x^2$.

Differentiating C with respect to x it is now easy to see that the cut size C is maximum when $\sum_{a \in S^{'}} a = x = T/2$ and the maximum cut size is $T.T/2 - (T/2)^2 = T^2/4$. Hence we have found a reduction from PARTITION to MAX CUT. As the parameter of both the problem instances are polynomially bounded, it is clearly a w-reduction. □

PERFECT CODE:

Input: A graph $G(V, E)$ with n vertices.

Task: Decide whether G has a Perfect Code $V^{'} \subseteq V$ (a set of vertices $V^{'} \subseteq V$ with the property that for each vertex $u \in V$ there is precisely one vertex in $N[u] \cap V^{'}$. $N[u]$ denotes the set containing vertex u and its neighbours, also known as closed neighbourhood.)

Parameter: $|V^{'}|.log(n)$

WEIGHTED EXACT CNF SATISFIABILITY:

Input: A formula ϕ in *CNF* of size m, and an integer k.

Task: Decide whether ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied.

Parameter: $k.log(m)$

Following relations can be proved between these two problems.

Proposition 11. WEIGHTED EXACT CNF SATISFIABILITY *and* PERFECT CODE *are w-reducible to each other.*

Proof idea: To prove the w-reduction from PERFECT CODE to WEIGHTED EXACT CNF SATISFIABILITY, we will take each vertex of PERFECT CODE instance G to be an input variable of WEIGHTED EXACT CNF SATISFIABILITY instance ϕ. Then

corresponding to vertices in closed neighbourhood of each vertex in the graph G, we will add a clause to ϕ. Each vertex in the closed neighbourhood will correspond to each literal in that clause. Clearly G has a perfect code of size k iff ϕ has weight k truth assignment that makes exactly one literal in each clause true. Clearly the reduction is a w-reduction. This simple reduction is mentioned in [43], Lemma 4.2. It is easy to see that same reduction works for our different parameter as well.

The reduction in the opposite direction is also mentioned by Downey and Fellows ([43], Lemma 4.3). Basically, to show the reduction from WEIGHTED EXACT CNF SATISFIABILITY to PERFECT CODE, we can use the same transformation used my them in the proof of Theorem 2.1 of [42]. (We have considered this transformation later in chapter 4 in more details). The parameters of both the problem instances are also polynomially bounded in each other and the result is proved. □

We are now going to consider the following problem.

EXACT HITTING SET:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$.

Task: Decide whether (V,E) has an exact hitting set $S \subseteq V$ (Exact hitting set is a subset $S \subseteq V$ such that $|S \cap e| = 1$ for all $e \in E$).

Parameter: $|S|.log(n)$

We now prove the following relation of this problem with EXACT COVER.

Proposition 12. EXACT COVER *and* EXACT HITTING SET *are w-reducible to each other.*

Proof. We use a very standard technique ([45]) to prove this result.

EXACT HITTING SET to EXACT COVER: To show the w-reduction from EXACT HITTING SET to EXACT COVER, let us consider a EXACT HITTING SET instance (V,E) and denote our reduced EXACT COVER instance as $(V^{'},E^{'})$. For each hyperedge e in E, we will take an element in $V^{'}$. Now we will construct a set corresponding to each vertex in the hypergraph (V,E), containing the elements corresponding to the hyperedges which contain that vertex. Collection of all those sets will be taken as $E^{'}$. So clearly $|V| = |E^{'}|$ and $|V^{'}| = |E|$. It is now easy to see that (V,E) has an Exact hitting set iff $(V^{'},E^{'})$ has an Exact cover. Parameter of both the problem instances are polynomially bounded in each other.

EXACT COVER to EXACT HITTING SET: Similarly, to show the w-reduction from EXACT COVER to EXACT HITTING SET, let us consider an EXACT COVER instance $(V^{'},E^{'})$ given to us and denote the EXACT HITTING SET instance that we

want to construct as (V,E). Corresponding to each set in $E^{'}$, we will construct a vertex in the hypergraph (V,E). Clearly $|V| = |E^{'}|$. Now we will add hyperedges to it in the following way. For each element in $V^{'}$ we will add a hyperedge connecting vertices corresponding to the sets from $E^{'}$ where the element is present. Hence, clearly $|V^{'}| = |E|$. It is now easy to see that (V,E) has an Exact hitting set iff $(V^{'},E^{'})$ has an Exact cover. Parameter of both the problem instances are polynomially bounded in each other.

Hence, EXACT COVER and EXACT HITTING SET are *w*-reducible to each other. □

We have already considered the CLIQUE problem. Now we are going to consider another variation of this same problem.

MULTICOLOURED CLIQUE:

Input: A graph $G = (V, E)$ with $|V| = n$, an integer k and a colouring function $c : V \rightarrow [k]$.

Task: Decide whether G has a multicoloured clique of size k (a clique containing exactly one vertex of each colour).

Parameter: $k.log(n)$

Following relation is already pointed out by Hermelin et. el. ([14], Lemma 10) where the explicit proof can be found in [38].

Proposition 13. CLIQUE *and* MULTICOLOURED CLIQUE *are w-reducible to each other.*

As CLIQUE is already proved to be VC_1-complete in Theorem 9, the above proposition also proves that MULTICOLOURED CLIQUE is VC_1-complete with respect to *w*-reduction. In the previously mentioned paper, Hermelin et. el. ([14], Lemma 11) also proved the following.

Proposition 14. MULTICOLOURED CLIQUE *is w-reducible to* EXACT HITTING SET.

In that result, although EXACT HITTING SET problem has different parameter, it is easy to observe that the same reduction works for our parameter as well. From the graph G, the MULTICOLOURED CLIQUE instance, they have constructed a EXACT HITTING SET instance (V,E), such that G has a multicoloured clique of size k iff (V,E) has an exact cover. Size of the exact cover in this reduction will be $\binom{k}{2}$.

We are now going to discuss the following result.

Proposition 15. PERFECT CODE *is w-reducible to* SUBSET SUM.

Proof idea: The reduction corresponding to this result is done by Downey and Fellows ([43], Lemma 4.4). We are briefly discussing that same reduction here which is working with our different parameter as well. Let $G(V,E)$ be a graph for which we are interested to know if it has a perfect code of size k. Without loss of generality, in this case we can consider that the vertex set of the graph $V = \{\ 0, \ldots (n-1)\ \}$. We take $L = \{x[i,j] \mid 1 \leq i \leq k, 0 \leq j \leq n-1\}$ and the positive integer B, where

$$x[i,j] = (k+1)^{n+k-i} + \sum_{u \in N[j]} (k+1)^u, B = \sum_{t=0}^{n+k-1} (k+1)^t$$

Here the objective is to construct the set of positive integers L such that L has a subset of size k summing up to B if and only if G has a perfect code of size k. Basically we can view that the numbers of L are represented in base $k+1$. In this way the correctness is easier to see. Parameter of both the problems are polynomially bounded as the number of vertices in G is polynomially bounded in number of elements in the set L. The proof idea is similar to what we have used to prove the w-reduction from EXACT COVER to SUBSET SUM. □

Proposition 16. INDEPENDENT SET *is w-reducible to* PERFECT CODE.

The reduction corresponding to the above proposition is shown by Downey and Fellows. Though the parameter in their work was k instead of $k.log(n)$, the same reduction works for us as well. Explicit proof can be found in [43] (Theorem 4.1).

We now define the following problem. We have already considered slightly different variation of this problem and placed it into VC_1 (k-CYCLE in section 3.2.1).

WEIGHTED K-CYCLE:

Input: A weighted graph G with m edges and n vertices, integer k and S.
Task: Decide whether there a cycle through k vertices of total cost S.
Parameter: $k.log(n)$

As we can see in the definition, in WEIGHTED K-CYCLE problem we try to find a proper ordering of k vertices in a graph $G(V,E)$, $|V| = n$, $|E| = m$, such that they construct a cycle and total weight is a given value, say S. Parameter is $k.log(n)$. A k-cycle consists of k vertices in order. Now, whether a given k vertices are forming k-cycle or not, can be easily verified in $poly(k.log(n))$ time. But to check if the total cost is S or not, we need to sum the weights. That may not be done in $poly(k.log(n))$

time. Hence, though we do not access the whole graph, we can not place this problem into VC_1 (same reason as SUBSET SUM, as explained in section 3.2.1). Now we can observe the following result. (Reduction originally shown in [43].)

Proposition 17. SUBSET SUM *is w-reducible to* WEIGHTED K-CYCLE.

Proof. To prove the reduction let us consider a SUBSET SUM instance with the set $S = \{x_1, x_2, \ldots, x_t\}$ and weight B. Now we construct a graph G as follows. For each x_i we construct two vertices a_i and b_i for the graph G. We join a_i to b_i with an edge of weight x_i. Let d is more than $\sum_{i=1}^{t} x_i$. We also join a_i to b_j when $i \neq j$ and give all such edges weight d. Now it is easy to see that the SUBSET SUM instance has a subset with k elements of total sum B iff G has a cycle with $2k$ vertices of total weight $b = B + kd$. Parameter for the SUBSET SUM instance is $k.log(t)$ and the WEIGHTED K-CYCLE instance is $2k.log(n)$. Clearly they are polynomially bounded to each other as the number of vertices in G, n is polynomially bounded with number of elements in set S, t. Hence the proof is done. □

We have already defined the problems MAXIMUM SUBGRAPH MATCHING and LARGEST COMMON SUBGRAPH (section 3.2.1). We are now going to see the following relation between them.

LARGEST COMMON SUBGRAPH:

Input: Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ and an integer k. Suppose n is the maximum of the numbers of vertices in these two graphs.

Task: Decide whether there exist subsets $V_1^{'} \subseteq V_1$ and $V_2^{'} \subseteq V_2$ with $|V_1^{'}| = |V_2^{'}| = k$ such that two vertex induced subgraphs $G^{'} = (V_1^{'}, E_1^{'})$ and $H^{'} = (V_2^{'}, E_2^{'})$ are isomorphic.

Parameter: $k.log(n)$

MAXIMUM SUBGRAPH MATCHING:

Input: Two directed graphs $G = (V_1, A_1)$ and $H = (V_2, A_2)$ and an integer k. Suppose n is the maximum of the numbers of vertices in these two graphs.

Task: Decide whether there is a subset $R \subseteq V_1 \times V_2$ with $|R| = k$ such that for all $\langle u, u^{'} \rangle$, $\langle v, v^{'} \rangle \in R$, $(u, v) \in A_1$ iff $(u^{'}, v^{'}) \in A_2$.

Parameter: $k.log(n)$

Proposition 18. LARGEST COMMON SUBGRAPH *is w-reducible to* MAXIMUM SUBGRAPH MATCHING *and both are in* VC_1.

From the definition it is easy to see that both the problems are quite similar, difference is, one problem is defined on directed graph and the other one is on undirected graph. So putting directions (both ways) in the edges we can easily w-reduce

a LARGEST COMMON SUBGRAPH problem instance to a MAXIMUM SUBGRAPH MATCHING instance.

The proof of membership in the VC_1 is easy as we can easily verify any witness in $poly(k.log(n))$ time. (the idea is similar to the way we have proved that SUB-GRAPH ISOMORPHISM $\in VC_1$, section 3.2.1.)

In this context we like to point out that the subgraphs defined in both the problems are actually vertex induced subgraphs. That is why just by removing directions, we can not reduce a MAXIMUM SUBGRAPH MATCHING instance to a LARGEST COMMON SUBGRAPH instance. But if the problems are defined in terms of edge induced subgraphs, reduction in that opposite direction works as well just by removing directions. We are not going to discuss that here. Rather, we are now going to consider the problem WEIGHTED EXACT CNF SATISFIABILITY again with different parameter.

WEIGHTED EXACT CNF SATISFIABILITY$_n$:

Input: A formula ϕ in *CNF* of size m over n variables, and an integer k ($\leq n$).

Task: Decide whether ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied.

Parameter: $k.log(n)$

From the definitions, it is trivial to see that the problem WEIGHTED EXACT CNF SATISFIABILITY is w-reducible to WEIGHTED EXACT CNF SATISFIABILITY$_n$. We are now going to prove the following.

Proposition 19. WEIGHTED EXACT CNF SATISFIABILITY$_n$ *and* EXACT HITTING SET *are w-reducible to each other.*

Proof. Reduction from EXACT HITTING SET to the problem WEIGHTED EXACT CNF SATISFIABILITY$_n$ is quite straight forward. Suppose we have a EXACT HITTING SET instance, hypergraph (V,E) ($|V| = n$, $|E| = m$). For each vertex in V, we simply take an input variable for our reduced formula ϕ in *CNF*. For each hyperedge e in E, we construct a clause taking conjunction of variables corresponding to the vertices in e. It is now easy to see that (V,E) has an exact hitting set of size k iff ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied. Clearly the parameters for both the problem instances are $k.log(n)$.

Now we have to prove the reduction in the opposite direction. To do that, we define a new problem as follows.

MONOTONE WEIGHTED EXACT CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another variable k ($\leq n$).
Task: Decide whether ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied.
Parameter: $k.log(n)$

Now we are going to show a w-reduction from the problem WEIGHTED EXACT CNF SATISFIABILITY$_n$ to MONOTONE WEIGHTED EXACT CNFSAT. This proof idea is inspired by the technique used by J. Flum and M. Grohe ([25], Lemma 7.5).

Let ϕ be a Boolean expression in conjunctive normal form consisting of l clauses $C_1, C_2, \ldots, C_l$ with variables $x_1, x_2, \ldots, x_n$. We are going to construct another Boolean formula $\phi^{'}$ from ϕ with $n^{'}$ (polynomially bounded in n) number of variables such that, ϕ is satisfiable with k number of variables assigned to be *True* with exactly one literal in each clause satisfied iff $\phi^{'}$ is also satisfiable with $k^{'}$ (polynomially bounded in k) number of variables assigned to be *True* with exactly one literal in each clause satisfied. In our construction of $\phi^{'}$, it will be conjunction of two different formulae. First formula (ψ) will not be dependent on ϕ, but the second one ($\psi^{'}$) will be. We have described the constructions of these two formulae below.

To construct such new formula, we introduce two sets of new variables $X_{i,j}$ (for $i \in [k]$ and $j \in [n]$) and $Y_{i,j,j^{'}}$ (for $i \in [k-1]$ and $1 \leq j < j^{'} \leq n$) with intending meanings as below:
$X_{i,j}$: the i^{th} variable set to be *True* is x_j,
$Y_{i,j,j^{'}}$: the i^{th} variable set to be *True* is x_j and the $(i+1)^{th}$ is $x_{j^{'}}$.

So total number of variables $n^{'}$ are bounded by $(k.n + (k-1)n(n-1)/2)$ which is clearly polynomially bounded in n as $k \leq n$.

We are now going to construct a new Boolean formula as conjunction of k Boolean formulae as follows:

$\psi = \phi_0 \wedge \phi_1 \wedge \ldots \wedge \phi_{k-1} \ldots (i)$

where $\phi_0 = (\bigwedge_{i=1}^{k} \bigvee_{a \in X_i} a) \wedge (\bigwedge_{i=1}^{k-1} \bigvee_{b \in Y_i} b)$, $X_i = \{X_{i,j} | j \in [n]\}$, $Y_i = \{Y_{i,j,j^{'}} | 1 \leq j < j^{'} \leq n\}$
$\phi_i = \bigwedge_{j \in [n]} (\bigvee_{1 \leq j_1 < j_2 \leq n, j_1 \neq j} (X_{i,j} \vee Y_{i,j_1,j_2}) \wedge \bigvee_{1 \leq j_1 < j_2 \leq n, j_2 \neq j} (X_{i+1,j} \vee Y_{i,j_1,j_2}))$, $i = 1, 2, \ldots, k-1$.

We claim that

Lemma 3. *Let l_1, l_2, …, $l_k \in [n]$ be k distinct integers. Any assignment of weight $(2k-1)$ satisfying ψ with exactly one literal in each clause satisfied and setting X_{1,l_1}, X_{2,l_2}, …, X_{k,l_k} to True, must set Y_{1,l_1,l_2}, Y_{2,l_2,l_3}, …, Y_{k-1,l_{k-1},l_k} to True.*

Proof. In ϕ_0, we can see there are exactly $2k-1$ clauses (corresponding to X_i, $1 \leq i \leq k$ and Y_i, $1 \leq i \leq k-1$). All of them are consists of different sets of input variables. So clearly exactly one variable in each X_i and Y_i should be *True* for any assignment of weight $(2k-1)$ satisfying ψ.

Let us now consider that X_{i,l_i} be the variable of X_i set to be *True*. For any such fixed $i \in [k-1]$, let $Y_{i,l,m}$ be the variable of Y_i set to *True*. If $l \neq l_i$, then in ϕ_i the sub-formula $\bigvee_{1 \leq j_1 < j_2 \leq n, j_1 \neq j}(X_{i,j} \vee Y_{i,j_1,j_2})$ will not be satisfied. Similarly if $m \neq l_{i+1}$ the sub-formula $\bigvee_{1 \leq j_1 < j_2 \leq n, j_2 \neq j}(X_{i+1,j} \vee Y_{i,j_1,j_2})$ will not be satisfied. Hence the lemma is proved. □

Now we will replace all the positive literals x_j of the initial formula ϕ by the following disjunction:
$\bigvee_{i \in [k]} X_{i,j}$
and all the negative literals $\bar{x_j}$ by the following formula (which is basically disjunctions of variables):
$\bigvee_{j_1 \in [n], j < j_1} X_{1,j_1} \vee (\bigvee_{i \in [k-1]} \bigvee_{j_1, j_2 \in [n], j_1 < j < j_2} Y_{i,j_1,j_2}) \vee \bigvee_{j_2 \in [n], j > j_2} X_{k,j_2} \ \ldots (ii)$
to construct the new Boolean formula $\psi^{'}$.

It is easy to see that x_j is *True* iff exactly one variable in the disjunction $\bigvee_{i \in [k]} X_{i,j}$ is *True*. But if x_j is *False*, i.e., $\bar{x_j}$ is *True*, x_j is either smaller (with respect to the ordering of the variables of ϕ by their indices, $x_1, x_2, \ldots, x_n$) than the first variable set to be *True* or between two consecutive variables set to be *True* or after the last variable set to be *True*. The formula (ii) above captures this.

We present $\phi^{'} = \psi \wedge \psi^{'}$ as our final formula. We can see that in $\phi^{'}$, all the literals are positive literals. Hence, it is a *Monotone* formula. From the above explanations we can understand that ϕ is satisfiable with k variables assigned to be *True* with exactly one literal in each clause satisfied iff $\phi^{'}$ is also satisfiable with $k^{'}$ ($k^{'} = 2k-1$) variables assigned to be *True* with exactly one literal in each clause satisfied. The parameter of the initial WEIGHTED EXACT CNF SATISFIABILITY$_n$ instance is $k.log(n)$ where as the parameter of the final instance is $(2k-1)log(n^{'})$. So clearly it is a w-reduction.

Remaining part of the proof is again quite simple. Suppose we have a MONOTONE WEIGHTED EXACT CNFSAT instance $\phi^{'}$ with $n^{'}$ input variables. For each variable of $\phi^{'}$, we construct a vertex of hypergraph (V,E), our final EXACT HITTING SET instance. For each clause C in $\phi^{'}$, we construct a hyperedge for (V,E) containing the vertices corresponding to the variables in C. It is now easy to see that (V,E) has an exact hitting set of size $k^{'}$ iff $\phi^{'}$ has a satisfying assignment of weight $k^{'}$ such that exactly

one literal in each clause is satisfied. Clearly the parameters for both the problem instances are $k^{'}.log(n^{'})$. Hence, we have finally proved that, WEIGHTED EXACT CNF SATISFIABILITY$_n$ and EXACT HITTING SET are *w*-reducible to each other. □

We have now seen all the *w*-reductions as presented in Figure 3.1. It is easy to see from the diagram, if either INDEPENDENT SET or CLIQUE is complete for VC_1, all the problems except BALANCED COMPLETE BIPARTITE SUBGRAPH in the Figure 3.1 are hard for VC_1.

In Theorem 9, we have already mentioned that CLIQUE is compression-complete for VC_1. Hence, we can conclude that all the above mentioned problems (except BALANCED COMPLETE BIPARTITE SUBGRAPH in the Figure 3.1), are hard for VC_1. In this context we like to discuss about the problem SUBSET SUM once again.

In section 3.2.1 we have mentioned why we can not place this problem into VC_1. In fact, we can not add k integers by a circuit of constant depth when the size of the integers are not constant. Hence, we can not place them in VC_t for any constant t. But we can always add k integers by a binary tree of depth $\Omega(log(k))$ where each node in the tree is adding two integers. We can add two binary integers by a circuit of constant depth (as integer addition is computable in AC^0). Circuit representation of each node do not need to be of constant size. Hence, it is easy to see that there is a straight forward way to put this SUBSET SUM problem into VC_K where size of the subset is k and $K = \Omega(log(k))$ (similar idea is used in [13], Section 2.6).

As can not place SUBSET SUM in VC_1, we can not place *Exact Cover* and EXACT HITTING SET in VC_1 using the *w*-reductions mentioned above where both the problems are *w*-reducible to SUBSET SUM. But we can see the following result.

Proposition 20. PERFECT CODE *is* VC_1*-complete.*

It is already mentioned that this problem is VC_1-hard. To show the completeness, we have to show that PERFECT CODE $\in VC_1$. For that purpose we use the result proved in [36]. But before that we need to define another problem.

NDTM HALTING:

Input: The code of a Turing machine M of size n, and an integer k.

Task: Decide whether M halts on the empty string in k steps.

Parameter: $k.log(n)$

Here the Turing machine may not have binary alphabet. Now we briefly discuss the proof idea.

Proof idea: Reduction technique from PERFECT CODE to NDTM HALTING is shown in [36] (Theorem 1). It is easy to see that this reduction can be directly used to claim that PERFECT CODE is *w*-reducible to NDTM HALTING. On the other hand, from the results in [14] (Theorem 3) it is easy to see that NDTM HALTING is VC_1-complete (it is better understandable from the detailed comparisons of different hierarchies in section 3.4). As all these above reductions are *w*-reductions, this result directly follows from it. □

As the parametric problems WEIGHTED EXACT CNF SATISFIABILITY and PERFECT CODE are *w*-reducible to each other (from Proposition 11) the above result implies that WEIGHTED EXACT CNF SATISFIABILITY is also VC_1-complete.

3.2.3 Problems in higher level of VC-hierarchy

We are now going to consider some of the problems in the higher level of VC-hierarchy and show some hardness and completeness results. We have already worked with the problem SET COVER$_{kn}$ before. Here we are considering the same problem with different parameter. Here is the definitions of the problems we are interested in next.

SET COVER:

Input: A set X with $|X| = n$, a family of sets S with $|S| = m$ where $S \subseteq \wp(X)$ ($\wp()$ denotes the power set) and an integer k.

Task: Decide whether there is a set cover of size k (a subset $S^* \subseteq S$ with $\bigcup S^* = X$).

Parameter: $k.log(m)$

HITTING SET:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$, and an integer k.

Task: Decide whether (V, E) has an hitting set of size k (a subset $S \subseteq V$ such that $|S| = k$ and $S \cap e \neq \phi$ for all $e \in E$).

Parameter: $k.log(n)$

We are now going to place these problems in some higher level of VC-hierarchy using the alternate definition of VC classification by *Verification* with *Preprocessing* (Definition 28). There are few similar examples mentioned in [13] (Section 2.6) for some other problems.

SET COVER $\in VC_3$: In SET COVER problem a set $X = \{x_1, x_2, \ldots, x_l\}$ and a collection S (of size m) of subsets of X is given to us. Question is, whether there is any sub-collection S^* of size k ($|S^*| = k$) of S such that each element in X is contained in at least one subset in S^*. Sub-collection S^* can be encoded in $k.log(m)$ bits. Hence the

witness of the problem, i.e., our parameter is $K = k.log(m)$. So for every witness in S^*, the verification algorithm tests that
$\forall x_i \in X \; \exists s \in S^* \; (\; x_i \in s \;)$.
Verification of $x_i \in s$ can be done in following way: $\exists y \in s \; (x_i = y)$. $\forall$ translates to *AND* gate over all x_i in X. Similarly $\exists$ translates to *OR* gate over inputs from S^* and s as shown above. Two consecutive *OR* is translated into one *OR* level. $(x_i = y)$ is tested by an *AND* over the bits of x_i and y. So the overall depth of the circuit is now 3 $(AND - OR - AND)$. Here the witness S^* can be encoded in size $poly(K)$. This section of the circuit above is taking witness w as input according to the definition. Remaining part of the circuit is taking the output after preprocessing according to the Definition 28 and is of size polynomially bounded in instance size. Clearly the overall size of the circuit is polynomially in the instance size as well (as all of X, x_i, s and S^* are polynomially bounded in input instance size). Hence, this NP parametric problem SET COVER belongs to VC_3. □

Similar to the SET COVER problem, we can consider HITTING SET to be defined in terms of set S and family of sets C and then place it in VC_3 as described below.

HITTING SET $\in VC_3$: In HITTING SET problem, we have a collection C of subsets of finite set S and a positive integer $k \leq |S|$. Question is, whether there is a subset $S^{'} \subseteq S$ with $|S^{'}| = k$ such that $S^{'}$ contains at least one element from each subset in C. Here $|C| = m$ and $|S| = n$. Clearly the parameter is $K = k.log(n)$. So the verification can be done in the following way:
$\forall S_i \in C \; \exists x_i \in S^{'} \; (x_i \in S_i)$.
Verification of $x_i \in S^{'}$ can be done in following way: $\exists y \in S^{'} \; (x_i = y)$. $\forall$ translates to *AND* gate over all S_i in C and $\exists$ translates to *OR* gate over all x_i and y as shown above. Two consecutive *OR* is translated into one *OR* level. $(x_i = y)$ is tested by an *AND* over the bits of x_i and y. So the overall depth is now 3 $(AND - OR - AND)$. Here $S^{'}$ can be encoded in size $poly(K)$. This section of the circuit above is taking witness w as input according to the definition. Remaining part of the circuit is taking the output after preprocessing according to the Definition 28 and is of size polynomially bounded in instance size. Clearly the overall size of the circuit is polynomially bounded in the instance size as well (as all of C, S_i, x_i and $S^{'}$ are polynomially bounded in input instance size). So this NP parametric problemHITTING SET belongs to VC_3. □

It can be proved easily that the HITTING SET and SET COVER problems are *w*-

reducible to each other. But we will come to the reductions later. We now define the following problem.

HITTING STRING:

Input: A finite set A of binary strings, each of same length n.

Task: Decide whether there is a string $x \in \{0,1\}^*$ with $|x| = n$ such that for each string $a \in A$ there is some j, $1 \leq j \leq n$, for which j^{th} symbol of a and the j^{th} symbol of x are identical.

Parameter: n

We can also prove the following for this problem.

HITTING STRING $\in VC_3$: We can see the definition of the HITTING STRING problem above. Clearly the parameter is $|x| = n$. So the verification can be done in the following way:

$\forall a \in A\ \exists j \in [1,n]\ (x_j = a_j)$.

$\forall$ translates to *AND* gate over all a in A and $\exists$ translates to *OR* gate over all j. $(x_j = a_j)$ is tested by an *AND* over the bits of x_j and a_j. So the overall depth of the circuit is now 3 ($AND - OR - AND$). Here each x_j is taking input from the witness x. This section of the circuit is taking witness w as input according to the definition. Remaining part of the circuit is taking the output after preprocessing according to the Definition 28 and is of size polynomially bounded in instance size. Clearly the overall size of the circuit is polynomially bounded in the instance size as well (as all of a, A and j are polynomially bounded in input instance size). So this NP parametric problem HITTING STRING belongs to VC_3. □

We now define the following problem.

LONGEST COMMON SUBSEQUENCE:

Input: A finite set A of strings over the alphabet $\{0,1\}$ and an integer k.

Task: Decide whether there is a common string of length k which is present in all the strings of A as a sub-string.

Parameter: k

LONGEST COMMON SUBSEQUENCE $\in VC_3$: In LONGEST COMMON SUBSEQUENCE problem, we have a finite set A of strings over the alphabet $\{0,1\}$. Let us consider that $|A| = m$. Question is, whether there is a common string of length k which is present in all the strings of A as a sub-string. Clearly k is the parameter. Suppose W (of size k) is given as the witness. So the verification can be done in the following way:

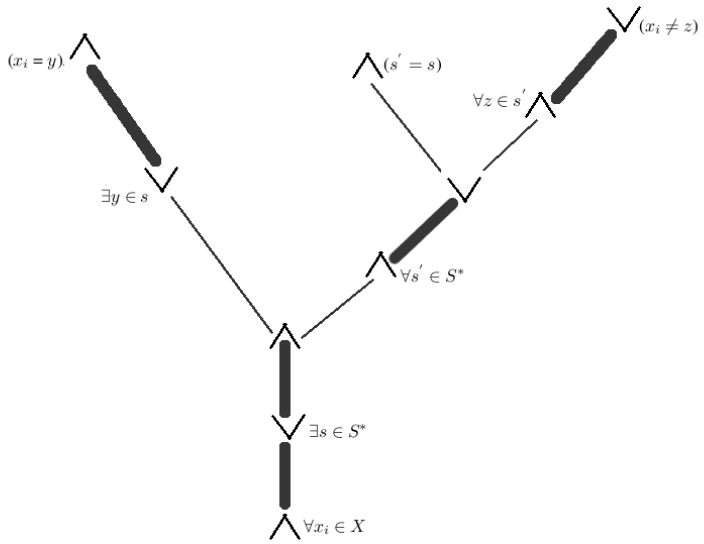
$\forall w_i \in A\ \exists v$ *contained in* w_i such that $\forall i \in |W|\ (v[i] = W[i])$ ($[i]$ denotes the value at i^{th} position).
$\forall$ translates to *AND* gate over all w_i and $[|W|]$ and $\exists$ translates to *OR* gate over all v. $(v[i] = W[i])$ is tested by an *AND* over the bits of $v[i]$ and $W[i]$. So the overall depth is now 3 ($AND-OR-AND$). Here in the circuit W can be encoded in size k. Remaining part of the circuit is taking the output after preprocessing according to the Definition 28 and is of size polynomially bounded in instance size. Clearly the overall size of the circuit is polynomially bounded in the instance size as well (as all of A, v and w_i are polynomially bounded in input instance size, total number of sub-strings from a given string is also polynomially bounded in the length of the string). So this NP parametric problem LONGEST COMMON SUBSEQUENCE belongs to VC_3. □

There are many such problems that can be placed in some level of VC hierarchy using the technique described above. We have just discussed some of them. But this approach gives just the upper bound of the problems in the hierarchy. To find the lower bound, we have to find the hardness results that we are going to discuss gradually. Just before finishing, we like to show how to place the problem *Exact Cover* in VC_6.

EXACT COVER $\in VC_6$: In *Exact Cover* problem a set $X = \{x_1, x_2, \ldots, x_n\}$ and a collection $\mathcal{S}$ ($|\mathcal{S}| = m$) of subsets of X is given to us. Question is, whether there is any sub-collection $\mathcal{S}^*$ of size k of $\mathcal{S}$ such that each element in X is contained in exactly one subset in $\mathcal{S}^*$. So the parameter is $K = k.log(m)$. The verification need to check whether each of the elements in X is assigned to a unique set in $\mathcal{S}^*$ or not. So the verification can be done in the following way:
$\forall x_i \in X\ \exists s \in \mathcal{S}^*\ [(x_i \in s) \bigwedge \forall s^{'} \in \mathcal{S}^*\ [(s^{'} = s) \bigvee (x_i \notin s^{'})]]$.

Verification of $x_i \in s$ can be done in following way: $\exists y \in s\ (x_i = y)$. Verification of $x_i \notin s^{'}$ can be done in following way: $\forall z \in s^{'}\ (x_i \neq z)$. $\forall$ translates to *AND* gate over all possible values of the variable z. Similarly $\exists$ translates to *OR* gate as shown in the diagram (Figure 3.2). $(x_i = y)$ is tested by an *AND* over the bits of x_i and y. Similarly we can test $(s^{'} = s)$. The circuit is shown in Figure 3.2. Thick lines corresponding to the *OR* and *AND* gates are for $\exists$ and $\forall$ respectively, as they are *AND*ing of *OR*ing over more than 2 inputs. Thin lines are used when we are taking *OR* or *AND* of exactly two inputs. All gates are marked below to understand which one is doing what. Unmarked gates are simply *AND* or *OR* of two inputs. So the overall depth considering alternating *AND* and *OR* gates is 6.

Figure 3.2: Verification circuit diagram for EXACT COVER



Here $\mathcal{S}^*$ can be encoded in size $poly(K)$. This section of the circuit above is taking witness w as input according to the definition. Remaining part of the circuit is taking the output after preprocessing according to the Definition 28 and is of size polynomially bounded in instance size. Clearly the overall size of the circuit is polynomially bounded in the instance size as well (as all of X, s, s' and $\mathcal{S}^*$ are polynomially bounded in input instance size). So this NP parametric problemEXACT COVER belongs to VC_6. □

We are now going to prove the following.

Theorem 10. HITTING STRING *is w-reducible to* HITTING SET.

Proof. Suppose we have a HITTING STRING instance A containing m binary strings, each of length n. Now for each string $s_i \in A$, $1 \leq i \leq m$, we will create following n elements $(\langle 0\rangle, s_i[0]), (\langle 1\rangle, s_i[1]), \ldots, (\langle n-1\rangle, s_i[n-1])$ and put them in a set S_i. Here $\langle j\rangle$ $(0 \leq j \leq n-1)$ denotes the binary encoding of j and $s_i[j]$ denotes the j^{th} bit of string s_i. We put all these S_i $(1 \leq i \leq m)$ into the set of hyperedges E for our reduced HITTING SET instance. We also include following n sets into E, $\{(\langle 0\rangle, 0), (\langle 0\rangle, 1)\}$, $\{(\langle 1\rangle, 0), (\langle 1\rangle, 1)\}$, …, $\{(\langle n-1\rangle, 0), (\langle n-1\rangle, 1)\}$. It is easy to see that all these lastly mentioned n sets are mutually exclusive. These n sets are constructed to ensure that exactly one element from each such set can contribute to a hitting set of size n.

We construct V, the set of vertices for our reduced HITTING STRING instance, as follows. $V = \{(\langle j\rangle, 0), (\langle j\rangle, 1) | 0 \leq j \leq n-1\}$. We can observe that $|E| = m+n$ and $|V|$

$= 2n$. It is now easy to see that the HITTING STRING instance A has a *hitting string* of size n iff (V,E) has a *hitting set* of size n. Parameter for our reduced HITTING SET instance instance is $n.log(2n)$, which is clearly polynomially bounded in n, the parameter for the HITTING STRING instance. Hence, the result is proved. □

We are now going to show some completeness and hardness results for VC_2. We start with the definitions of the following two problems.

NAE-SAT:

Input: A formula ϕ in *CNF* with n variables.

Task: Decide whether ϕ has a satisfying assignments such that each clause of ϕ contains at least one *True* and one *False* literal.

Parameter: n

SET SPLITTING:

Input: A set S of n elements and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S.

Task: Decide whether there exists a partition of the set S into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the sets in $\mathcal{C}$ is contained in either of s_1 and s_2.

Parameter: n

Now we prove the hardness result in the next lemma.

Lemma 4. SET SPLITTING *is compression-hard for* VC_2.

Proof. We will show a w-reduction from SAT to SET SPLITTING to prove this result as SAT is VC_2-complete from the definition ([13]). Let us now consider that a SAT instance ϕ over the variables $x_1, x_2, \ldots, x_n$ is given to us. Let us also consider that c_1, c_2, …, c_l are the clauses of ϕ. Now we consider a separate variable y, other than x_1, x_2, …, x_n. We modify each clause c_i as $c_i^{'} = c_i \vee \neg y$ and introduce a new clause $c^{'} = y$. Now our modified formula $\phi^{'} = c_1^{'} \wedge c_2^{'} \wedge \ldots \wedge c_l^{'} \wedge c^{'}$. It is now easy to see that to make $\phi^{'}$ satisfiable y must be assigned to be *True*. So each of the clauses $c_1^{'}$, $c_2^{'}$, …, $c_l^{'}$ will contain at least one literal (i.e., $\neg y$) which is assigned to be *False*. Hence the clauses $c_1^{'}$, $c_2^{'}$, …, $c_l^{'}$ will satisfy the property of NAE-SAT but not the clause $c^{'}$. We can easily understand that ϕ is satisfiable iff $\phi^{'}$ satisfiable and the reduction is w-reduction. Because the parameter of the reduced instance, the number of variables is increased by just one compared to the parameter of the given instance.

Now we consider the following construction of SET SPLITTING instance from $\phi^{'}$. We construct a set S containing all the variables in the formula $\phi^{'}$ and their negations.

This S is the set to be split for our reduced SET SPLITTING instance. For each variable-negation pair $((y, \neg y), (x_1, \neg x_1), (x_2, \neg x_2), \ldots, (x_n, \neg x_n))$ we make a subset. We also construct a subset for each clause of $\phi^{'}$, containing elements corresponding to the literals present into that clause, except the clause $c^{'}$.

Now, if $\phi^{'}$ has a satisfying assignment, we will put all the literals assigned to be *True* in one set (say s_1) and all the literals assigned to be *False* into other (say s_2). It is now easy to see that it is a valid set splitting where each subset contains at least one element from both the sets s_1 and s_2. Now suppose, S can be split in the required way. We take one part of the split set to represent *True* literals, the other *False* literals. We will make the part containing y to be *True*. We can check that each subset must contain one *True* element and one *False* element, which is exactly what the satisfiability problem requires for the clauses $c_1^{'}, c_2^{'}, \ldots, c_l^{'}$. The clause $c^{'}$ will be satisfiable as we have chosen the part containing y to be *True*. Hence there exists a SET SPLITTING for S iff $\phi^{'}$ is satisfiable iff ϕ is satisfiable. Here the size of the parameter for SET SPLITTING is $(2n+2)$ which is polynomially bounded by the parameter size of the initial SAT problem ϕ. So clearly the reduction described above is a w-reduction. Hence SET SPLITTING is compression-hard for VC_2. □

Now we are going to discuss the following completeness result.

Theorem 11. SET SPLITTING *is VC_2-complete.*

Proof. We have already shown the hardness result for the SET SPLITTING problem in lemma 4. Now we are going to show that SET SPLITTING $\in VC_2$ to prove this theorem. We have already seen the problem definition. Parameter of the problem is n, the number of elements in the set S. The containment of this problem in VC_2 will be shown in two steps. Firstly, w-reduction from SET SPLITTING to NAE-SAT and then w-reduction from NAE-SAT to SAT.

Let us consider a SET SPLITTING instance (S,C). We now represent each $S_i \in C$ as a clause of a NAE-SAT instance ϕ, containing its members as literals. So the elements, $x_1, x_2, \ldots, x_n$ are the variables here. Now we can easily observe that there exists a Set Splitting for (S,C) iff ϕ is satisfiable with NAE-SAT property (because variables in one split set will be assigned to be *True* and the other to be *False*). Besides, the parameter size is also preserved. So clearly it is a w-reduction.

For proving the next part, let us consider an NAE-SAT instance ϕ. Now for each clause c_i we will construct another clause $c_i^{'}$ in the following way: for each literal $l \in c_i$ we put $\neg l$ in $c_i^{'}$. Now inserting all the new clauses into ϕ we will construct a

new formula ψ. So ψ is twice in size with respect to ϕ. Both of them have the same variables. We can easily see that ψ is satisfiable iff ϕ is *NAE*-satisfiable. So clearly this is a *w*-reduction.

Hence we have proved that SET SPLITTING $\in VC_2$ and the theorem follows. □

We are working with the NAE-SAT problem for quite some time. It is now the time to prove the completeness result for this problem. We have already shown a *w*-reduction from NAE-SAT to SAT above in the final part of the proof for Theorem 11. It proves that NAE-SAT $\in VC_2$ as SAT is VC_2-complete from the definition ([13]). Besides, we have already shown a *w*-reduction from SET SPLITTING to NAE-SAT above in the first part of the proof for Theorem 11. As SET SPLITTING is VC_2-complete from the above theorem, the above mentioned reduction proves that NAE-SAT is VC_2-hard. Combining them we can state the following theorem.

Theorem 12. NAE-SAT *is* VC_2*-complete.*

We can find a direct reduction from SAT to NAE-SAT to show the hardness result for NAE-SAT. We have used this technique later in chapter 4 (Theorem 25).

Theorem 13. SAT *w-reducible to* NAE-SAT.

Proof. Let us now consider a SAT instance ϕ over input variables $x_1, x_2, \ldots, x_n$. Let us also consider that $c_1, c_2, \ldots, c_l$ are the clauses of ϕ. Now we consider two separate variables y and z, other than $x_1, x_2, \ldots, x_n$. We modify each clause c_i as $c_i^{'} = c_i \vee \neg y$ and introduce a new clause $c^{'} = y \vee z$. Now our modified formula $\phi^{'} = c_1^{'} \wedge c_2^{'} \wedge \ldots \wedge c_l^{'} \wedge c^{'}$. Now we are going to prove that ϕ is satisfiable iff $\phi^{'}$ is *NAE*-satisfiable.

Suppose ϕ is satisfiable. Now for any satisfying assignments to $x_1, x_2, \ldots, x_n$ for ϕ, we will use the same assignments for those variables in $\phi^{'}$. Next, we will assign $y = 1$ and $z = 0$. We can now see that $\phi^{'}$ is *NAE*-satisfiable. So for any truth assignments for ϕ there exists a truth assignments for $\phi^{'}$ satisfying NAE-SAT property.

Now suppose $\phi^{'}$ is *NAE*-satisfiable. So obviously either $y = 1$ and $z = 0$ or $y = 0$ and $z = 1$. If $y = 1$, we will use the same assignments of $x_1, x_2, \ldots, x_n$ from $\phi^{'}$ to ϕ. If $y = 0$, we will invert the assignments to all the variables in $\phi^{'}$. We can understand that still it is *NAE*-satisfiable. Now we will use the same procedure to assign $x_1, x_2, \ldots, x_n$ for ϕ as now $y = 1$. Clearly we have found a satisfying assignment for ϕ.

Here we can see that the parameter, the number of variables (for both the problems number of variables are the parameters), is increased by just two for the reduced

instance. So the above reduction is *w*-reduction. Hence, SAT *w*-reducible to NAE-SAT. □

We are now going to consider a slightly different variation of the SET SPLITTING problem.

SET SPLITTING$_k$:

Input: A set S of n elements and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S.

Task: Decide whether there exists a partition of the set S into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the sets in $\mathcal{C}$ is contained in either of s_1 and s_2.

Parameter: $min\{|s_1|, |s_2|\}.log(n)$

We prove the following hardness result for this problem.

Theorem 14. SET SPLITTING$_k$ *is* VC_2*-hard.*

It is easy to see that SET SPLITTING is trivially *w*-reducible to SET SPLITTING$_k$ as any witness of second problem instance can be encoded in bits polynomially bounded in the parameter for the first problem instance. As SET SPLITTING is VC_2-complete (Theorem 11), the above argument shows that SET SPLITTING$_k$ is VC_2-hard.

The reduction used in lemma 4 can also be used to prove that SAT is *w*-reducible to SET SPLITTING$_k$. The only difference is, the parameter for the final reduced instance will be $(n+1).log(2n+2)$ instead of $2n+2$. As SAT is VC_2-complete from the definition, in this way we can get an alternate proof for the above theorem. We can also find a direct reduction from NAE-SAT to SET SPLITTING$_k$.

Theorem 15. NAE-SAT *is w-reducible to* SET SPLITTING$_k$

Proof. Let us consider that a NAE-SAT instance ϕ over variables $x_1, x_2, \ldots, x_n$ is given to us. We now consider the following construction of SET SPLITTING$_k$ instance from ϕ. We construct a set S containing all the variables in the NAE-SAT formula ϕ and their negations. This S is the set to be split for our reduced SET SPLITTING$_k$ instance. For each variable-negation pair $((x_1, \neg x_1), (x_2, \neg x_2), \ldots, (x_n, \neg x_n))$ we construct a subset. We also construct a subset for each clause, containing elements corresponding to the literals present into that clause. We can observe that the set-to-be-split S can be split in the desired way if and only if the formula ϕ has a required satisfying assignment. Because, one part of the split set represents *True* literals, the other *False* literals. Hence, we can check that each subset must contain at least one *True* element and one *False* element, which is exactly what this NAE-SAT problem requires. Here the size

of the parameter for the reduced SET SPLITTING$_k$ instance is $n.log(2n)$ which is polynomially bounded in n, the parameter of the initial NAE-SAT problem instance. So clearly the reduction described above is a w-reduction. □

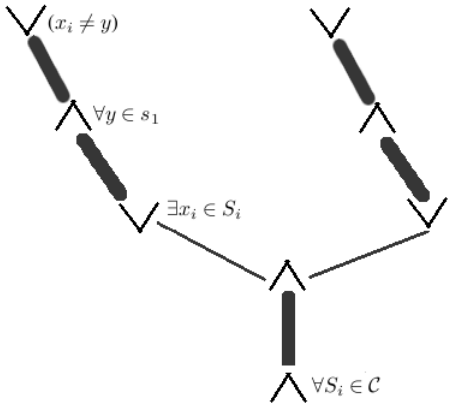
The reduction used above can also be used to show a direct w-reduction from NAE-SAT to SET SPLITTING.

Now, we are going to use the technique of verification algorithm once again to place the problem SET SPLITTING$_k$ into VC_4.

SET SPLITTING$_k$ $\in VC_4$: In SET SPLITTING problem a set $S = \{x_1, x_2, \ldots, x_n\}$ and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S is given to us. Question is, whether there exists a partition of the set S into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the subsets in $\mathcal{C}$ is contained in either of s_1 and s_2. Let us take our parameter $k = min\{|s_1|, |s_2|\}.log(n)$. Now, the verification can be done in the following way:
$\forall S_i \in \mathcal{C}$ [($S_i \nsubseteq s_1$) $\bigwedge$ ($S_i \nsubseteq s_2$)]
The part ($S_i \nsubseteq s_1$) can be re-written as:
$\exists x_i \in S_i \; \forall y \in s_1 \; (x_i \neq y)$.
$\forall$ translates to *AND* gate over all possible values of the variable y. Similarly $\exists$ translates to *OR* gate. $(x_i \neq y)$ can be tested by *OR* gate as shown in the diagram (Figure 3.3). So the above part will give a circuit of depth 3 as shown in the diagram. Similarly we can re-write the section ($S_i \nsubseteq s_2$). Final $\forall$ will correspond to another *AND* gate over all possible values of S_i. Thick lines corresponding to the *OR* and *AND* gates are for $\exists$ and $\forall$ respectively, as they are *AND*ing of *OR*ing over more than 2 inputs. Thin lines are used when we are taking *OR* or *AND* of exactly two variables. So the overall depth considering alternating *AND* and *OR* gates is 4.
As all the terms C, s_1, s_2 and S_i, used in the verification circuit are bounded polynomially in instance size, the over all size of the circuit is polynomially bounded in the size of the input problem instance.

Here in the circuit, minimum of s_1 and s_2 can be encoded in size $poly(k)$. This section of the circuit above is taking witness w as input according to the definition. Remaining part of the circuit is taking the output after preprocessing according to the Definition 28 and is of size polynomially bounded in instance size. Hence, this NP problem belongs to VC_4. □

Figure 3.3: Verification circuit diagram for SET SPLITTING$_k$



SET SPLITTING, the same same problem as above but with larger parameter, is already proved to be VC_2-complete. Same problem with smaller parameter intuitively looks more difficult to compression, though we can not prove anything specific for this particular problem. But one can definitely try to prove that SET SPLITTING$_k \in VC_3$ to minimize the gap.

We are now going to define another quite popular parametric problem.

WEIGHTED CNF SATISFIABILITY:

Input: A formula ϕ in *CNF* with n variables, and an integer k.

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

Proposition 21. WEIGHTED CNF SATISFIABILITY *is* VC_2*-hard.*

Proof. SAT is already known to be VC_2 complete [13]. So we are going to show a w-reduction from SAT to WEIGHTED CNF SATISFIABILITY to prove the hardness result. Let us consider a SAT instance ϕ with n variables. The parameter is n. Let us consider $x_1, x_2, \ldots, x_n$ to be those n variables of ϕ. Now we introduce another n variables $y_1, y_2, \ldots, y_n$ such that $y_1 = \neg x_1$, $y_2 = \neg x_2$, $\ldots$, $y_n = \neg x_n$. Now we replace all the negative literals of ϕ by corresponding y variable. After that we add following clauses to the formula: $(y_i \vee x_i) \wedge (\neg y_i \vee \neg x_i)$ for all $i = 1, 2, \ldots, n$ to ensure the above mentioned relations between x and y variables. Let us denote the resulting formula by ϕ'. Clearly the number of variables of ϕ' is $2n$. Now we take the weight k of the problem WEIGHTED CNF SATISFIABILITY to be n. It is now easy to see that ϕ is satisfiable iff

ϕ' is satisfiable with an assignment that assigns exactly k (= n) of its $2n$ variables the Boolean value 1. The parameter for the WEIGHTED CNF SATISFIABILITY problem is $k.log(2n) = n.log(2n)$. So the reduction mentioned above is a w-reduction. Hence WEIGHTED CNF SATISFIABILITY is VC_2-hard. □

We have already defined SET COVER and HITTING SET before and placed them into VC_3. Now we are going to consider them again and show the next result.

Proposition 22. SET COVER *and* HITTING SET *are w-reducible to each other.*

Proof. We will show this result by using the well known reductions between these two problems (similar idea is already used to proof the Proposition 12). We are going to see that those reductions are actually w-reduction.

HITTING SET to SET COVER: To show the w-reduction from HITTING SET to SET COVER, let us consider a HITTING SET instance (V,E) and denote our reduced SET COVER instance as (V',E'). For each hyperedge e in E, we will take an element in V'. Now we will construct a set corresponding to each vertex in the hypergraph (V,E), containing the elements corresponding to the hyperedges which contain that vertex. Collection of all those sets will be taken as E'. So clearly $|V| = |E'|$ and $|V'| = |E|$. It is now easy to see that (V,E) has a hitting set of size k iff (V',E') has a set cover of size k. Parameter of both the problem instances are polynomially bounded in each other.

SET COVER to HITTING SET: Similarly, to show the w-reduction from SET COVER to HITTING SET, let us consider an SET COVER instance (V',E') given to us and denote the HITTING SET instance that we want to construct as (V,E). Corresponding to each set in E', we will construct a vertex in the hypergraph (V,E). Clearly $|V| = |E'|$. Now we will add hyperedges to it in the following way. For each element in V' we will add a hyperedge connecting vertices corresponding to the sets from E' where the element is present. Hence, clearly $|V'| = |E|$. It is now easy to see that (V,E) has a hitting set of size k iff (V',E') has a set cover of size k. Parameter of both the problem instances are polynomially bounded in each other.

Hence, SET COVER and HITTING SET are w-reducible to each other. □

We now define the following problem.

DOMINATING SET:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a dominating set of size k (a set D of k vertices for which every vertex not in D has a neighbour in D).

Parameter: $k.log(n)$

Proposition 23. DOMINATING SET *is w-reducible to* SET COVER.

This proof is very much straight forward. For each vertex of a DOMINATING SET instance G, we will take an element in the set X for the reduced SET COVER instance (X,S). Now for each vertex, we will construct a subset containing all the elements corresponding to the neighbouring vertices as well as that vertex itself. Collection of all those subsets will be taken as S. Clearly there is a dominating set of size k for G iff there is a set cover of size k for (X,S).

In [13], it is already shown that DOMINATING SET is in VC_3. We are now going to show the next result.

Proposition 24. HITTING SET *and* WEIGHTED CNF SATISFIABILITY *are w-reducible to each other.*

Proof. This idea behind this proof is quite similar to what we have used in the proof of the Proposition 19. Suppose we have a HITTING SET instance, hypergraph (V,E) ($|V| = n$, $|E| = m$). For each vertex in V, we simply take an input variable for our reduced formula ϕ in CNF. For each hyperedge e in E, we construct a clause taking conjunction of variables corresponding to the vertices in e. It is now easy to see that (V,E) has a hitting set of size k iff ϕ has a satisfying assignment such that k variables are assigned to be *True*). Clearly the parameters for both the problem instances are $k.log(n)$.

Now to show the reduction in the opposite direction we will define the following problem.

MONOTONE WEIGHTED CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another variable k ($\leq n$).

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

Now, we can show that there is a w-reduction from WEIGHTED CNF SATISFIABILITY to MONOTONE WEIGHTED CNFSAT. The idea behind this proof is inspired by the technique used by J. Flum and M. Grohe ([25], Lemma 7.5). We have already used that technique in the proof of Proposition 19 few times back. Exactly same reduction works here as well. (We will review this technique in detail once again in chapter 4.)

Remaining part of the proof is again quite simple. Suppose we have a MONOTONE WEIGHTED CNFSAT instance $\phi^{'}$ with $n^{'}$ input variables. For each variable of $\phi^{'}$, we construct a vertex of hypergraph (V,E), our final EXACT HITTING SET instance. For each clause C in $\phi^{'}$, we construct a hyperedge for (V,E) containing the vertices corresponding to the variables in C. It is now easy to see that (V,E) has a hitting set of size $k^{'}$ iff $\phi^{'}$ has a satisfying assignment of weight $k^{'}$. Clearly the parameters for both the problem instances are $k^{'}.log(n^{'})$. Hence, we have finally proved that, HITTING SET and WEIGHTED CNF SATISFIABILITY are *w*-reducible to each other. □

Downey and Fellows [42] (Theorem 2.1) has already shown a reduction from the problem WEIGHTED CNF SATISFIABILITY to DOMINATING SET, but that reduction is not *w*-reduction. Because, in that construction, number of vertices of the reduced graph will depend on number of clauses of the initial formula. We will discuss about that construction in more details later as that can be used to find a direct *w*-reduction from WEIGHTED CNF SATISFIABILITY to SET COVER.

We are now going to define the following parametric problem which is very popular in the domain of VLSI design and testing.

MINIMUM TEST SET:

Input: A set S with $|S| = n$, another set $T \subseteq Pair(S)$ where $Pair(S)$ denotes all the distinct pairs of elements from S, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ ($\wp()$ denotes the power set), and an integer k.

Task: Decide whether there is a subset $C^{'} \subseteq C$ with $|C^{'}| = k$, such that for every distinct pair u and v from T, there exists a set c in $C^{'}$ such that either of u and v is present in c, but not both.

Parameter: $k.log(m)$

We now prove the following.

Proposition 25. MINIMUM TEST SET *and* SET COVER *are w-reducible to each other.*

Proof. We are now going to show *w*-reductions from MINIMUM TEST SET to SET COVER and vice versa.

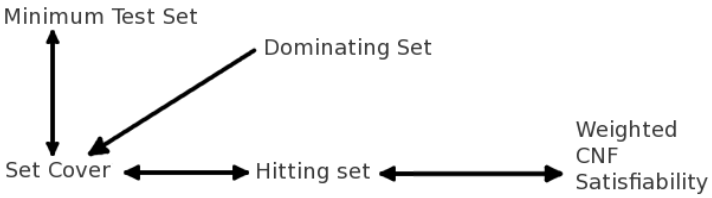
We first show a *w*-reduction from MINIMUM TEST SET to SET COVER. For that, we are going to consider all the distinct pairs of elements from T of the given MINIMUM TEST SET instance (S,T,C). We take this set of pairs as V for our reduced SET COVER problem instance (V,E). So the number of elements in V is at most $n(n-1)/2$ when $|S| = n$. Now for each set C_i in C we construct a set E_i for the family of sets E. E_i is containing elements from V corresponding to all (u,v) from T such that either

of u and v is in C_i but not both. So the number of sets of E will be m when $|C| = m$. Now it is easy to see that the MINIMUM TEST SET instance has a k-element test sets iff the SET COVER instance has a k-element set cover. Parameter of both the problem instances are clearly polynomially bounded with each other.

We are now going to show the w-reduction from SET COVER to MINIMUM TEST SET. For that, we take an element x which is not present in V for our given SET COVER instance (V, E). Now we take S for our reduced MINIMUM TEST SET problem instance (S, T, C) as follows: $S = V \cup \{x\}$. We take $T = \bigcup_{v_i \in V} \{v_i, x\}$. We take C of the MINIMUM TEST SET problem instance to be same as E of the given SET COVER problem instance. So the number of elements of C will be m when $|E| = m$. Now it is easy to see that the MINIMUM TEST SET instance has a k-element test sets iff the SET COVER instance has a k-element set cover. Parameter of both the problem instances are clearly polynomially bounded with each other.

Hence, MINIMUM TEST SET and SET COVER are w-reducible to each other. □

We have seen some w-reductions recently. Let us represent them in a diagram as follows.



In the diagram above, arrow from the problem A to B implies that there is a w-reduction from problem A to B that we have considered. Now combining Proposition 21, 22, 24 and 25 we can see that (also clear from the diagram) WEIGHTED CNF SATISFIABILITY, HITTING SET, MINIMUM TEST SET and SET COVER are VC_2-hard and belong to VC_3.

3.3 Introduction to new complexity class VC_E

We are now going to define a new complexity class with respect to a very popular parametric problem that we have considered already, but with different parameter.

EXACT CNF-SAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables.

Task: Is there any satisfying assignment for ϕ such that exactly one literal in each

clause of ϕ is assigned to be *True* ?

Parameter: n

We are now defining a complexity class VC_E as follows:

Definition 30. *VC_E is the class of parametric problems which are w-reducible to* EXACT CNF-SAT.

Harnik and Naor ([13]) have already pointed out that there are many natural problems present in VC_1 and it is always interesting to further classify this VC_1. Introduction of this new class is primarily motivated by this. We are going to prove that VC_E is a subset of VC_1. Hence, in this way we can further classify VC_1. Besides, in section 3.4 we are going to see that weighted and un-weighted version of satisfiability problems are in different level of hierarchies. From Proposition 11 and 20 we already know that the problem WEIGHTED EXACT CNF SATISFIABILITY is VC_1-complete. So, it is quite natural to define a new complexity class with respect to the un-weighted version of the same problem.

Theorem 16. $VC_E \subseteq VC_1$

To prove Theorem 16, we will show a *w*-reduction from EXACT CNF-SAT to WEIGHTED EXACT CNF SATISFIABILITY.

Proof. Let us consider a EXACT CNF-SAT instance ϕ with n variables. The parameter is n. Let us consider $x_1, x_2, \ldots, x_n$ to be those n variables of ϕ. Now we introduce another n variables $y_1, y_2, \ldots, y_n$ such that $y_1 = \neg x_1$, $y_2 = \neg x_2$, $\ldots$, $y_n = \neg x_n$. Now we replace all the negative literals of ϕ by corresponding y variable. Then we add following clauses to the formula: $(y_i \vee x_i) \wedge (\neg y_i \vee \neg x_i)$ for all $i = 1, \ldots, n$ to ensure the above mentioned relations between x and y variables. Let us denote the resulting formula by ϕ'. Clearly the number of variables of ϕ' is $2n$. Now we take the weight k of WEIGHTED EXACT CNF SATISFIABILITY problem to be n. It is now easy to see that ϕ is satisfiable with exactly one *True* literal in each clause iff ϕ' is satisfiable with an assignment that assigns exactly k of its $2n$ variables the Boolean value 1 and makes exactly one *True* literal in each clause of ϕ'. If the size of ϕ' is m, the parameter for the reduced instance is $k.log(m) = n.log(m)$. As $log(m)$ is polynomially bounded in n, the parameter of the input EXACT CNF-SAT instance ϕ, the reduction mentioned above is a *w*-reduction. Hence, $VC_E \subseteq VC_1$. □

We are now going to define some of the interesting decision problems and then going to show that, they are eventually, VC_E-complete.

EXACT HITTING SET$_n$:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$.

Task: Decide whether (V, E) has an exact hitting set (a subset $S \subseteq V$ such that $|S \cap e| = 1$ for all $e \in E$).

Parameter: n

EXACT COVER$_n$:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set).

Task: Decide whether there is an exact cover (a subset $S \subseteq E$ of pairwise disjoint sets with $\bigcup S = V$).

Parameter: m

Theorem 17. EXACT HITTING SET$_n$ *and* EXACT COVER$_n$ *are* VC_E*-complete.*

Proof. We will prove this result by showing a series of reductions. We have already used similar techniques to prove some different results in previous sections.

EXACT CNF-SAT to EXACT HITTING SET$_n$: Suppose EXACT CNF-SAT instance ϕ is given to us with n variables and l clauses. Corresponding to each variable and its negation, we introduce one element in V for our reduced EXACT HITTING SET$_n$ problem instance (V, E). So, there will be $2n$ elements in V. Corresponding to each clause, we introduce one set in E which contains elements from V corresponding to the literals in that clause. So each clause contribute to one set in E of our reduced EXACT HITTING SET$_n$ problem instance. Besides, we introduce n new sets, containing x_i and $\bar{x_i}$ in one set, for each $i = 1, 2, \ldots, n$. All these new sets, along with the sets corresponding to all the clauses, form the hyperedge set E of our reduced EXACT HITTING SET$_n$ problem instance. It is now easy to see, ϕ is exactly satisfiable, iff (V, E) has a exact hitting set. The parameter of the reduced problem $2n$ is also polynomially bounded with respect to the initial one. Hence, it is a w-reduction.

EXACT HITTING SET$_n$ to EXACT CNF-SAT: This reduction is quite straight forward. Suppose we have a EXACT HITTING SET$_n$ instance, hypergraph (V, E) ($|V| = n$, $|E| = m$). For each vertex in V, we simply take an input variable for our reduced formula ϕ in *CNF*. For each hyperedge e in E, we construct a clause taking conjunction of variables corresponding to the vertices in e. It is now easy to see that (V, E) has an exact hitting set iff ϕ has a satisfying assignment such that exactly one literal in each clause is *True*. Clearly the parameters for both the problem instances are n and hence, it is a w-reduction.

EXACT HITTING SET$_n$ to EXACT COVER$_n$: To show the w-reduction from EXACT HITTING SET$_n$ to EXACT COVER$_n$, let us consider a EXACT HITTING SET$_n$ instance (V,E) and denote our reduced EXACT COVER$_n$ instance as $(V^{'},E^{'})$. For each hyperedge e in E, we will take an element in $V^{'}$. Now we will construct a set corresponding to each vertex in the hypergraph (V,E), containing the elements corresponding to the hyperedges which contain that vertex. Collection of all those sets will be taken as $E^{'}$. So clearly $|V| = |E^{'}|$ and $|V^{'}| = |E|$. It is now easy to see that (V,E) has an Exact hitting set iff $(V^{'},E^{'})$ has an Exact cover. Parameter of both the problem instances are polynomially bounded in each other.

EXACT COVER$_n$ to EXACT HITTING SET$_n$: Similarly, to show the w-reduction from EXACT COVER$_n$ to EXACT HITTING SET$_n$, let us consider an EXACT COVER$_n$ instance $(V^{'},E^{'})$ given to us and denote the EXACT HITTING SET$_n$ instance that we want to construct as (V,E). Corresponding to each set in $E^{'}$, we will construct a vertex in the hypergraph (V,E). Clearly $|V| = |E^{'}|$. Now we will add hyperedges to it in the following way. For each element in $V^{'}$ we will add a hyperedge connecting vertices corresponding to the sets from $E^{'}$ where the element is present. Hence, clearly $|V^{'}| = |E|$. It is now easy to see that (V,E) has an Exact hitting set iff $(V^{'},E^{'})$ has an Exact cover. Parameter of both the problem instances are polynomially bounded in each other.

From the above reductions, it can be concluded that both EXACT COVER$_n$ and EXACT HITTING SET$_n$ are VC_E-complete. □

We are now going to define another interesting problem as follows to prove the next completeness result.

EXACT TEST SET:

Input: A set S with $|S| = n$, another set $T \subseteq Pair(S)$ where $Pair(S)$ denotes all the distinct pairs of elements from S, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ ($\wp()$ denotes the power set).

Task: Decide whether there is a subset $C^{'} \subseteq C$, such that for every distinct pair u and v from T, there exists exactly one set c in $C^{'}$ such that either of u and v is present in c, but not both.

Parameter: m

Theorem 18. EXACT TEST SET *is* VC_E*-complete.*

Proof. We will prove this result but showing w-reductions from EXACT TEST SET to EXACT COVER$_n$ and vice versa.

***w*-reduction from EXACT TEST SET to EXACT COVER$_n$:** Firstly, we are going to consider all the distinct pairs of elements from T of the given EXACT TEST SET instance (S,T,C). We take this set of pairs as V for our reduced EXACT COVER$_n$ problem instance (V,E). So the number of elements in V is at most $n(n-1)/2$ when $|S| = n$. Now for each set C_i in C we construct a set E_i for the family of sets E. E_i is containing elements from V corresponding to all (u,v) from T such that either of u and v is in C_i but not both. So the number of sets of E will be m when $|C| = m$. Now it is easy to see that the EXACT TEST SET instance (S,T,C) has an exact test sets iff the EXACT COVER$_n$ instance (V,E) has an exact cover. Parameter of both the problem instances are clearly polynomially bounded with each other.

***w*-reduction from EXACT COVER$_n$ to EXACT TEST SET:** Now, we take an element x which is not present in V for our given EXACT COVER$_n$ instance (V,E). Now we take S for our reduced EXACT TEST SET problem instance (S,T,C) as follows: $S = V \cup \{x\}$. We take $T = \bigcup_{v_i \in V} \{v_i, x\}$. We take C of the EXACT TEST SET problem instance to be same as E of the given EXACT COVER$_n$ problem instance. So the number of elements of C will be m when $|E| = m$. Now it is easy to see that the EXACT TEST SET instance (S,T,C) has an exact test sets iff the EXACT COVER$_n$ instance (V,E) has an exact cover. Parameter of both the problem instances are clearly polynomially bounded with each other.

From the above reductions, it can be concluded that EXACT TEST SET is VC_E-complete. □

As $VC_E \subseteq VC_2$, it is easy to understand that there is some *w*-reduction from EXACT CNF-SAT to SAT. Now we are going to prove a direct *w*-reduction from EXACT CNF-SAT to SAT. The trick used to prove this result is very simple and similar things are used later to prove some more interesting results (chapter 4).

Proposition 26. EXACT CNF-SAT *is w-reducible to* SAT.

Proof. To prove this result, we consider a EXACT CNF-SAT instance ϕ (in conjunctive normal form) with n variables (x_1, x_2, …, x_n) and l clauses. We assume that its clauses are $C_0, C_1, \ldots, C_{l-1}$. We also consider that

$C_i = \bigvee_{j=0}^{i_t-1} l_{i,j}$, $0 \leq i \leq l-1$.

It means, for $0 \leq i \leq l-1$, any clause C_i is disjunctions of i_t literals.

Now we construct $C_i^{'}$ corresponding to each C_i, $0 \leq i \leq l-1$ as follows.

$C_i^{'} = C_i \wedge \bigwedge_{0 \leq p < q \leq (i_t-1)} (\bar{l_{i,p}} \vee \bar{l_{i,q}})$.

It is easy to see that $C_i^{'}$ is satisfiable iff exactly one literal in C_i is assigned to be *True*.

Now we construct the new formula $\phi^{'}$ as follows.

$\phi^{'} = \bigwedge_{i=0}^{l-1} C_i^{'}$

It is easy to see that $\phi^{'}$ is satisfiable iff exactly one literal in each clause of ϕ is assigned to be *True*. The number of variables (parameter) in both ϕ and $\phi^{'}$ are same. Hence, EXACT CNF-SAT is *w*-reducible to SAT. □

3.4 Comparison with existing hierarchy

There are some other related hierarchies that we are going to consider now. Let us start with the definitions of these existing hierarchies. But before that, we consider the following definitions.

For $t \geq 0$ and $d \geq 1$, we inductively define the following classes $\Gamma_{t,d}$ and $\Delta_{t,d}$ of formulas following (as defined in [25]):

$\Gamma_{0,d} := \{\lambda_1 \wedge \ldots \wedge \lambda_c : c \in [d]$ and $\lambda_1, \ldots, \lambda_c$ are literals $\}$,

$\Delta_{0,d} := \{\lambda_1 \vee \ldots \vee \lambda_c : c \in [d]$ and $\lambda_1, \ldots, \lambda_c$ are literals $\}$,

$\Gamma_{t+1,d} := \{\bigwedge_{i \in I} \delta_i$: I is a finite non-empty index set and $\delta_i \in \Delta_{t,d}$ for all $i \in I\}$,

$\Delta_{t+1,d} := \{\bigvee_{i \in I} \delta_i$: I is a finite non-empty index set and $\delta_i \in \Gamma_{t,d}$ for all $i \in I\}$.

From the definitions above we can see that, $\Gamma_{1,3}$ is actually the set of all the 3-*CNF* formulae, and $\Gamma_{2,1}$ is nothing but the set of formulae in *CNF*. Given a class of Boolean formula Φ, we consider Φ^+, $\Phi^- \subseteq \Phi$ to denote the restrictions of Φ to formulas containing only positive and negative literals, respectively (as introduced in [14]). For any given Φ, let us consider the following two parametric problems:

Definition 31. Φ-*WSAT*$(k.log(n))$*:*

Input*: A formula $\phi \in \Phi$ over n input variables and an integer k ($\leq$ n).*

Membership*: Decide whether ϕ has a satisfying assignment of weight k.*

Parameter*: $k.log(n)$*

Definition 32. Φ-*SAT*(n) *:*

Input*: A formula $\phi \in \Phi$ over n input variables.*

Membership*: Decide whether ϕ has a satisfying assignment.*

Parameter*: n*

In particular, we will be focusing in Γ-$WSAT(k.log(n))$ and Γ-$SAT(n)$ to see the definitions of the following hierarchies as defined in [14].

Definition 33. *Let $t \geq 1$ be an integer. The classes $WK[t]$ and $MK[t]$ are defined by*
– $WK[t]$:= All the parametric problems w-reducible to $\bigcup_{d\in\mathbb{N}}[\Gamma_{t,d}$-$WSAT(k.log(n))]$.
– $MK[t]$:= All the parametric problems w-reducible to $\bigcup_{d\in\mathbb{N}}[\Gamma_{t,d}$-$SAT(n)]$.

The relation between WK and MK hierarchies as shown in [14]:

Theorem 19. $MK[1] \subseteq WK[1] \subseteq MK[2] \subseteq WK[2] \subseteq MK[3] \subseteq \ldots \subseteq FPT$.

In the original definitions for WK and MK hierarchies ([14]), there is no restriction in the parameter choice for the problems. But in our VC-hierarchy the parameters can be interpreted as the *witness size* for some natural NTM deciding the language. So if we now restrict the parameters to be natural witness length in both the WK and MK hierarchies, we can compare them with VC hierarchy as below. In the very first place we can point out that, when parameters can be interpreted as the *witness size* for some natural NTM deciding the language,

- $MK[1]$ is the class containing the languages having polynomial size kernel. Hence, it is same as our VC_0.
- From the proof of Proposition 6 and Theorem 5, [14], we can see that $WK[1]$ and VC_1 are equivalent.

Let us now try to find the relation between $MK[t]$ and VC_t for $t \geq 2$. For that let us consider the following definition.

Definition 34. DEPTH$_k$FORMULASAT*: For any $k \geq 2$ consider the parametric problem called* DEPTH$_k$FORMULASAT*:*
***Input**: A formula ϕ of size m and depth at most k over n variables.*
***Membership**: $\phi \in$ DEPTH$_k$FORMULASAT if there exists a satisfying assignment to ϕ.*
***Parameter**: n*

In [13], it is already pointed out, how without loss of generality we can assume that the top level Boolean operation for the given formula in the above definition can be taken as *AND* operation. It is also easy to observe that SAT is VC_2-complete (as pointed out in [13]) as any depth 2 circuit is equivalent to *CNF* formula. From

the Lemma 5, [14], we can eventually see that the problem DEPTH$_k$FORMULASAT is complete for $MK[t]$. But for constant depth, we can see that it is equivalent to DEPTH$_k$CIRCUITSAT (as for any out-degree more than one in the circuit, we can copy that sub-circuit constant number of times to make sure that all the out-degrees in the circuit is one, satisfying the property of a formula). So $MK[t] = VC_t$, for all $t \geq 2$.

Hence, restricting the parameter as the length of a natural witness and combining Theorem 8 and 19 with above arguments, we can say,

Theorem 20. $MK[1] = VC_0 \subseteq WK[1] = VC_1 \subseteq MK[2] = VC_2 \subseteq WK[2] \subseteq MK[3] = VC_3 \subseteq \ldots \subseteq FPT$.

From the above relations we can see that, weighted (to which WK hierarchy is based on) and un-weighted (to which VC hierarchy is based on) variations of same satisfiability problems are in different alternate levels with respect to the difficulty in their instance compression.

Important Observation:
SET COVER (as well as HITTING SET) is already proved to be $WK[2]$-complete in [14]. We have proved that these problems are VC_2-hard and present in VC_3 (= $MK[3]$). From the relation mentioned above (specifically, $VC_2 \subseteq WK[2] \subseteq VC_3 = MK[3]$), we can see that WEIGHTED CNF SATISFIABILITY, HITTING SET, MINIMUM TEST SET and SET COVER are not VC_3-complete unless WK and MK hierarchies coincide with each other at level 3 of MK-hierarchy. Besides, DOMINATING SET is also proved to be $WK[2]$-complete in [14] together with WEIGHTED CNF SATISFIABILITY, HITTING SET, and SET COVER. It implies that all these above mentioned problems (WEIGHTED CNF SATISFIABILITY, HITTING SET, and SET COVER) are w-reducible to DOMINATING SET, which we could not show before. On the other hand, we can see that MINIMUM TEST SET is $WK[2]$-complete which is not shown in [14]. In this way, work for either of the hierarchies can help to find new results for all of them.

3.5 Appendix: Definitions of the NP parametric problems we have considered

NP languages in parametric form:

ANTIMONOTONE WEIGHTED 2-CNF SATISFIABILITY

Input: A formula ϕ in 2-*CNF* (each clause contains 2 literals) with n input variables where all the literals in ϕ are negative literals, and an integer k.

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

BALANCED COMPLETE BIPARTITE SUBGRAPH:

Input: A Bipartite graphs $G = (V, E)$ with n vertices and an positive integer $k \leq |V|$.

Task: Decide whether there are two disjoint independent sets $V_1, V_2 \subseteq V$ such that $|V_1| = |V_2| = k$ and $u \in V_1$, $v \in V_2$ implies that $\{u, v\} \in E$.

Parameter: $k.log(n)$

BINARY NDTM HALTING:

Input: The code of a Turing machine M of size n with a binary alphabet, and an integer k.

Task: Decide whether M halts on the empty string in k steps.

Parameter: $k.log(n)$

CLIQUE:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a clique of size k (a pairwise adjacent subset of k vertices).

Parameter: $k.log(n)$

DEPTH$_k$CIRCUITSAT:

Input: A circuit C of size m and depth at most k over n variables, $k \geq 2$.

Task: Decide if there exists a satisfying assignment to C.

Parameter: n

DEPTH$_k$FORMULASAT:

Input: A formula ϕ of size m and depth at most k over n variables, $k \geq 2$.

Task: Decide if there exists a satisfying assignment to ϕ.

Parameter: n

DOMINATING SET:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a dominating set of size k (a set D of k vertices for which

every vertex not in D has a neighbour in D).

Parameter: $k.log(n)$

EXACT CNF-SAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables.

Task: Is there any satisfying assignment for ϕ such that exactly one literal in each clause of ϕ is assigned to be *True* ?

Parameter: n

EXACT COVER:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set).

Task: Decide whether there is an exact cover $S \subseteq E$ for V (Exact cover is a subset $S \subseteq E$ of pairwise disjoint sets with $\bigcup S = V$).

Parameter: $|S|.log(m)$

EXACT COVER$_n$:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set).

Task: Decide whether there is an exact cover (a subset $S \subseteq E$ of pairwise disjoint sets with $\bigcup S = V$).

Parameter: m

EXACT HITTING SET:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$.

Task: Decide whether (V, E) has an exact hitting set $S \subseteq V$ (Exact hitting set is a subset $S \subseteq V$ such that $|S \cap e| = 1$ for all $e \in E$).

Parameter: $|S|.log(n)$

EXACT HITTING SET$_n$:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$.

Task: Decide whether (V, E) has an exact hitting set (a subset $S \subseteq V$ such that $|S \cap e| = 1$ for all $e \in E$).

Parameter: n

EXACT TEST SET:

Input: A set S with $|S| = n$, another set $T \subseteq Pair(S)$ where $Pair(S)$ denotes all the distinct pairs of elements from S, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ ($\wp()$ denotes the power set).

Task: Decide whether there is a subset $C' \subseteq C$, such that for every distinct pair u and v from T, there exists exactly one set c in C' such that either of u and v is present in c,

but not both.

Parameter: m

GRAPH ISOMORPHISM:

Input: Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ both with n vertices.

Task: Decide whether G and H are isomorphic to each other.

Parameter: n

HITTING SET:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$, and an integer k.

Task: Decide whether (V, E) has an hitting set of size k (a subset $S \subseteq V$ such that $|S| = k$ and $S \cap e \neq \phi$ for all $e \in E$).

Parameter: $k.log(n)$

HITTING STRING:

Input: A finite set A of binary strings, each of same length n.

Task: Decide whether there is a string $x \in \{0,1\}^*$ with $|x| = n$ such that for each string $a \in A$ there is some j, $1 \leq j \leq n$, for which j^{th} symbol of a and the j^{th} symbol of x are identical.

Parameter: n

INDEPENDENT SET:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has an independent set of size k (a pairwise non-adjacent subset of k vertices).

Parameter: $k.log(n)$

k-COLOURABILITY PROBLEM:

Input: A graph $G(V,E)$ with n vertices, and an integer k.

Task: Decide whether G is k colourable.

Parameter: $n.log(k)$

k-CYCLE:

Input: A graph G with n vertices, and an integer k.

Task: Decide whether G has a cycle of length k.

Parameter: $k.log(n)$

LARGEST COMMON SUBGRAPH:

Input: Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ and an integer k. Suppose n is the maximum of the numbers of vertices in these two graphs.

Task: Decide whether there exist subsets $V_1^{'} \subseteq V_1$ and $V_2^{'} \subseteq V_2$ with $|V_1^{'}| = |V_2^{'}| = k$ such that two vertex induced subgraphs $G^{'} = (V_1^{'}, E_1^{'})$ and $H^{'} = (V_2^{'}, E_2^{'})$ are isomorphic.

Parameter: $k.log(n)$

LocalCircuitSAT:

Input: A string x of length m and a circuit C over ($k + k.log\ m$) variables and of size $O(\ k + k.log\ m\ )$.

Task: Decide whether there exists a list I of k locations in x such that $C(x(I), I) = 1$.

Parameter: $k + k.log\ m$

Longest Common Subsequence:

Input: A finite set A of strings over the alphabet $\{0, 1\}$ and an integer k.

Task: Decide whether there is a common string of length k which is present in all the strings of A as a sub-string.

Parameter: k

Max Cut:

Input: A weighted graph $G(V, E)$ with m edges and n vertices, an integer B.

Task: Decide whether there exists a subset $V^{'} \subseteq V$ such that the total size of the cut (sum of weights of edges between vertex sets $V^{'}$ and $V - V^{'}$) is $\geq B$.

Parameter: $|V^{'}|.log(n)$

Maximum Subgraph Matching:

Input: Two directed graphs $G = (V_1, A_1)$ and $H = (V_2, A_2)$ and an integer k. Suppose n is the maximum of the numbers of vertices in these two graphs.

Task: Decide whether there is a subset $R \subseteq V_1 \times V_2$ with $|R| = k$ such that for all $\langle u, u^{'} \rangle$, $\langle v, v^{'} \rangle \in R$, $(u, v) \in A_1$ iff $(u^{'}, v^{'}) \in A_2$.

Parameter: $k.log(n)$

Minimum Test Set:

Input: A set S with $|S| = n$, another set $T \subseteq Pair(S)$ where $Pair(S)$ denotes all the distinct pairs of elements from S, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ ($\wp()$ denotes the power set), and an integer k.

Task: Decide whether there is a subset $C^{'} \subseteq C$ with $|C^{'}| = k$, such that for every distinct pair u and v from T, there exists a set c in $C^{'}$ such that either of u and v is present in c, but not both.

Parameter: $k.log(m)$

Monotone Weighted CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another variable k ($\leq n$).

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

MONOTONE WEIGHTED EXACT CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another variable k ($\leq n$).

Task: Decide whether ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied.

Parameter: $k.log(n)$

MULTICOLOURED CLIQUE:

Input: A graph $G = (V, E)$ with $|V| = n$, an integer k and a colouring function $c : V \rightarrow [k]$.

Task: Decide whether G has a multicoloured clique of size k (a clique containing exactly one vertex of each colour).

Parameter: $k.log(n)$

MULTICOLOURED WEIGHTED CNF SATISFIABILITY:

Input: A formula ϕ in *CNF* with n variables, an integer k and a colouring function $c : X \rightarrow [k]$.

Task: Decide whether ϕ is satisfiable by an multicoloured assignment of Hamming weight k (an assignment where no two variables of same colour are assigned to 1).

Parameter: $k.log(n)$

NAE-SAT:

Input: A formula ϕ in *CNF* with n variables.

Task: Decide whether ϕ has a satisfying assignments such that each clause of ϕ contains at least one *True* and one *False* literal.

Parameter: n

NDTM HALTING:

Input: The code of a Turing machine M of size n, and an integer k.

Task: Decide whether M halts on the empty string in k steps.

Parameter: $k.log(n)$

PARTITION:

Input: A set S of m non-negative integers.

Task: Decide whether there is a set $S^{'} \subseteq S$ such that, $\sum_{a \in S'} a = \sum_{a \in S - S'} a$.

Parameter: $(min(|S^{'}|, |S - S^{'}|)).log(m)$

PERFECT CODE:

Input: A graph $G(V, E)$ with n vertices.

Task: Decide whether G has an Perfect Code $V^{'} \subseteq V$ (a set of vertices $V^{'} \subseteq V$ with

the property that for each vertex $u \in V$ there is precisely one vertex in $N[u] \cap V^{'}$. $N[u]$ denotes the set containing vertex u and its neighbours, also known as closed neighbourhood.)

Parameter: $|V^{'}|.log(n)$

SAT:

Input: A formula ϕ in CNF with n variables.

Task: Decide whether ϕ is satisfiable.

Parameter: n

SET COVER:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set) and an integer k.

Task: Decide whether there is a set cover of size k (a subset $S \subseteq E$ with $\bigcup S = V$).

Parameter: $k.log(m)$

SET COVER$_{kn}$:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set) and an integer k.

Task: Decide whether there is a set cover of size k (a subset $S \subseteq E$ with $\bigcup S = V$).

Parameter: kn

SET PACKING:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set) and an integer k.

Task: Decide whether there is a set packing of size k (a subset $S \subseteq E$ such that, for all distinct pair of sets $S_1, S_2 \in S$, S_1 and S_2 are mutually exclusive).

Parameter: $k.log(m)$

SET SPLITTING:

Input: A set S of n elements and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S.

Task: Decide whether there exists a partition of the set S into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the sets in $\mathcal{C}$ is contained in either of s_1 and s_2.

Parameter: n

SET SPLITTING$_k$:

Input: A set S of n elements and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S.

Task: Decide whether there exists a partition of the set S into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the sets in $\mathcal{C}$ is contained in either of s_1 and s_2.

Parameter: $min\{|s_1|, |s_2|\}.log(n)$

SUB-GRAPH ISOMORPHISM:

Input: Two graphs $G = (V_1, E_1)$ with n vertices, and $H = (V_2, E_2)$ with k vertices.

Task: Decide whether G contains a subgraph isomorphic to H (a subset $V \subseteq V_1$ and a subset $E \subseteq E_1$ such that $|V| = |V_2|$, $|E| = |E_2|$, and there exist a one-to-one function f: $|V_2| \rightarrow |V|$ satisfying $\{u, v\} \in E_2$ iff $\{f(u), f(v)\} \in E$).

Parameter: $k.log(n)$

SUBSET SUM:

Input: A set S of m non-negative integers, and another integer B.

Task: Decide whether there is a subset $S^{'} \subseteq S$ such that the summation of all the elements is $S^{'}$ is B.

Parameter: $|S^{'}|.log(m)$

VERTEX COVER:

Input: A graph $G(V, E)$ with n vertices, and an integer k.

Task: Decide if there is a vertex cover (a set $D \subseteq V$, such that for every edge $\{u, v\} \in E$, either u or v are in D) of size k in G.

Parameter: $k.log(n)$

WEIGHTED 3-CNF SATISFIABILITY

Input: A formula ϕ in 3-*CNF* (each clause contains at most 3 literals) with n input variables, and an integer k.

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

WEIGHTED CNF SATISFIABILITY:

Input: A formula ϕ in *CNF* with n variables, and an integer k.

Task: Decide whether ϕ is satisfiable by an assignment of Hamming weight k (an assignment that assigns exactly k variables the Boolean value 1).

Parameter: $k.log(n)$

WEIGHTED EXACT CNF SATISFIABILITY:

Input: A formula ϕ in *CNF* of size m, and an integer k.

Task: Decide whether ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied.

Parameter: $k.log(m)$

WEIGHTED EXACT CNF SATISFIABILITY$_n$:

Input: A formula ϕ in *CNF* of size m over n variables, and an integer k ($\leq n$).

Task: Decide whether ϕ has a satisfying assignment of weight k (k variables are assigned to be *True*) such that exactly one literal in each clause is satisfied.

Parameter: $k.log(n)$

WEIGHTED K-CYCLE:

Input: A weighted graph G with m edges and n vertices, integer k and S.

Task: Decide whether there a cycle through k vertices of total cost S.

Parameter: $k.log(n)$

Chapter 4

Counting hierarchy with respect to Compression

We have considered several parametric problems so far. They are basically decision problems. That means, answer to those problems are either *True* or *False*. In this chapter we have defined parametric counting problems and a counting hierarchy depending on the notion of instance compression. After the #P-completeness results for Permanent by L. G. Valiant [33], counting problems gained a lot of interests in the field of complexity theory. We have considered them more formally in parametric form and defined classification of them with respect to compression. We have extended both VC and WK hierarchy here for counting versions and compared them. Other than the classification of some of the interesting counting problems in hierarchical complexity classes, we have also considered a large varieties of circuit satisfiability problems (e.g. weighted monotone satisfiability, exact satisfiability etc.) and studied their behaviour with respect to this notion of compression.

4.1 Definitions and other important notions

Definition 35. ***Parametric relation:*** *A parametric relation is a subset of* $\{ \langle x, y, 1^n \rangle \mid x \in \{0,1\}^*, y \in \{0,1\}^n, n \in \mathbb{N} \}$. *The term n is known as the parameter and y is known as a witness of the problem.*

Definition 36. ***Parametric counting problem:*** *Suppose* $R \subseteq \{ \langle x, y, 1^n \rangle \mid x \in \{0,1\}^*, y \in \{0,1\}^n, n \in \mathbb{N} \}$ *is a parametric relation. A parametric counting problem corresponding to R is a function* $F : \{0,1\}^* \times \mathbb{N} \to \mathbb{N}$ *such that* $F(x,n) = |\{y \mid \langle x,y,1^n \rangle \in R\}|$.

Similarly, we can define *Parametric relation* and *Parametric counting problem* corresponding to any alphabet Σ, not just $\{0,1\}$.

A *Parametric counting problem* F is said to be in $\#P$ if corresponding parametric relation R is polynomially computable in size of x.

Definition 37. *Let $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ and $G : \Pi^* \times \mathbb{N} \to \mathbb{N}$ be parametric counting problems.*
A weakly parametric w-parsimonious reduction from F to G consists of a pair of polynomial time computable functions $\sigma : \Sigma^* \times \mathbb{N} \to \Pi^* \times \mathbb{N}$ and $\tau : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ such that $F(x,n) = \tau(x, G(\sigma(x,n)))$ and if $(y,l) = \sigma(x,n)$, l is polynomially bounded in n.

A parametric w-parsimonious reduction is called strong if $F(x,n) = G(\sigma(x,n))$ (i.e., $\tau(x, G(\sigma(x,n))) = G(\sigma(x,n))$).

We write $F \leq^{w}_{pars} G$ to denote that there is a strongly parametric w-parsimonious reduction from F to G. The weaker version is denoted by $\leq^{w'}_{pars}$.

Definition 38. *Let $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ and $G : \Pi^* \times \mathbb{N} \to \mathbb{N}$ be parametric counting problems.*
A parametric w-T-reduction from F to G is an algorithm with an oracle for G that solves any instance (x,n) of F in time polynomially bounded in $|x|$ in such a way that for all oracle queries the instances (y,l) satisfy the condition that l is polynomially bounded in n.

We write $F \leq^{w}_{T} G$ to denote that there is a parametric w-T-reduction from F to G.

Analogous to *weakly parametric w-parsimonious reduction*, we can interpret *parametric w-T-reduction* from F to G by pair of algorithms (σ, τ). The difference for *parametric w-T-reduction* is, in this case the algorithm τ (which is now the algorithm with an oracle for G as in Definition 38) can make queries to oracle for G multiple times (polynomial in input problem size). Besides, for all such oracle queries for G, the instances (y,l) satisfy the condition that l is polynomially bounded in n where n is the parameter of the input problem instance. Thus, it is easy to see that if $F \leq^{w'}_{pars} G$ or $F \leq^{w}_{pars} G$, then $F \leq^{w}_{T} G$. Hence we can summarize the relations among different type of counting reductions as follows,

$F \leq^{w}_{pars} G \Rightarrow F \leq^{w'}_{pars} G$ and $F \leq^{w'}_{pars} G \Rightarrow F \leq^{w}_{T} G$

Definition 39. #DEPTH$_t$CIRCUITSAT*: For any $t \geq 2$ consider the counting problem called* #DEPTH$_t$CIRCUITSAT*:*
***Input**: A circuit C of size m and depth at most t over n variables.*

***Parameter**: n*

***Task**: How many satisfying assignments are there for C ?*

Definition 40. #LOCALCIRCUITSAT*:*

***Input**: A string x of length m and a circuit C over (n + n.log m) variables and of size O(n + n.log m).*

***Parameter**: (n + n.log m)*

***Task**: How many lists I of n locations in x is there such that $C(\langle x(I), I\rangle) = 1$?*

Definition 41. ***Weakly-Parsimonious Compression for counting problem:*** *Any #P parametric counting problem $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ is called Weakly-Parsimonious compressible if there are polynomial-time computable functions $f : \Sigma^* \times \mathbb{N} \to \Pi^* \times \mathbb{N}$ and $\tau : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, such that for each $x \in \Sigma^*$ and $n \in \mathbb{N}$, if $f(x,n) = (y,l)$, both $|y|$ and l are polynomially bounded in n and $F(x,n) = \tau(x, G(f(x,n)))$ for some parametric counting problems $G : \Pi^* \times \mathbb{N} \to \mathbb{N}$.*

Definition 42. ***Turing Compression for counting problem:*** *Any #P parametric counting problem $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ is called Turing compressible if there is a polynomial-time computable algorithm with an oracle for G for some counting problem $G : \Pi^* \times \mathbb{N} \to \mathbb{N}$ that can solve any instance (x,n) of F in time polynomially bounded in $|x|$ in such a way that for all oracle queries, the instances (y,l) satisfy the condition that both y and l are polynomially bounded in n.*

Similarly, it is easy to see that if any counting problem is Weakly-Parsimonious compressible, it is also Turing compressible. But the opposite may not be true.

Theorem 21. *If there is a parametric w-T-reduction from F to G and G is Turing compressible, F is also Turing compressible.*

Proof. To prove this result we will describe a Turing compression algorithm for the counting problem F. Suppose that A is the algorithm with an oracle for G that can solve any F instance (x,n). So A is the w-T-reduction from F to G. We also assume that B is the Turing compression algorithm with an oracle for H for some counting problem H, that can solve any instance (y,l) of G in time polynomially bounded in $|y|$ in such a way that for all oracle queries, the instances (z,k) satisfy the condition that both z and k are polynomially bounded in l. Hence, B is the Turing compression algorithm for G.

We are now going to describe another compression algorithm with an oracle for H for some counting problem H which will solve any instance (x,n) of F in time

polynomially bounded in $|x|$ in such a way that for all oracle queries, the instances (z,k) satisfy the condition that both z and k are polynomially bounded in n. This compression algorithm will work as follows. To solve any (x,n) of F, this algorithm will follow the steps of algorithm A. Whenever algorithm A needs to make an oracle query for a G instance (y,l), our compression algorithm will follow the steps of algorithm B to solve the G instance (y,l). To solve this G instance (y,l), similar to algorithm B, our compression algorithm will also make oracle quarries (oracle for H for some counting problem H) such that for all those oracle queries the instances (z,k) satisfy the condition that both z and k are polynomially bounded in l. As A is a w-T-reduction from F to G, we know that l is polynomially bounded in n and also $|y|$ is polynomially bounded in $|x|$. Hence, both z and k are polynomially bounded in n. So, our algorithm with an oracle for H can solve any instance (x,n) of F where for all the queries for the instances (z,k), both z and k are polynomially bounded in n. As runtime of algorithm A is polynomially bounded in $|x|$ and algorithm B is polynomially bounded in $|y|$, runtime of our compression algorithm is also polynomially bounded in $|x|$ (as $|y|$ is polynomially bounded in $|x|$). Hence, F is Turing compressible and the result is proved. □

From the above proof, it is also easy to understand that If there is a parametric w-T-reduction from F to G and another parametric w-T-reduction from G to H, there is a parametric w-T-reduction from F to H. Similarly, we can also prove that, if there is a weakly parametric w-parsimonious reduction from F to G and G is Weakly-Parsimonious compressible, F is also Weakly-Parsimonious compressible.

Definition 43. *$\#VC_0$ is the class of those #P parametric counting problems $F : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ which are Turing compressible.*

We are now going to introduce a new hierarchy for parametric counting problems.

Definition 44. *For $t \geq 2$, $\#VC_t$ is the class of counting problems that are parametric w-T-reducible to* #DEPTH$_t$CIRCUITSAT. *$\#VC_1$ is the class of all parametric counting problems that are parametric w-T-reducible to* #LOCALCIRCUITSAT.

Similar to its decision counterpart, we can also prove that,

Theorem 22. $\#VC_0 \subseteq \#VC_1 \subseteq \#VC_2 \subseteq \ldots \subseteq \#VC_{poly(m)}$.

The original proof for VC-hierarchy ([13]) is strongly parsimonious and work for analogous counting problems as well.

We are now going to define the counting hierarchy corresponding to another existing hierarchy.

For $t \geq 0$ and $d \geq 1$, we inductively define the following classes $\Gamma_{t,d}$ and $\Delta_{t,d}$ of formulas following [25]:

$\Gamma_{0,d} := \{\lambda_1 \wedge \ldots \wedge \lambda_c : c \in [d]$ and $\lambda_1, \ldots, \lambda_c$ are literals $\}$,

$\Delta_{0,d} := \{\lambda_1 \vee \ldots \vee \lambda_c : c \in [d]$ and $\lambda_1, \ldots, \lambda_c$ are literals $\}$,

$\Gamma_{t+1,d} := \{\bigwedge_{i \in I} \delta_i$: I is a finite non-empty index set and $\delta_i \in \Delta_{t,d}$ for all $i \in I\}$,

$\Delta_{t+1,d} := \{\bigvee_{i \in I} \delta_i$: I is a finite non-empty index set and $\delta_i \in \Gamma_{t,d}$ for all $i \in I\}$.

Thus, it is easy to understand that $\Gamma_{1,3}$ is the set of all the 3-*CNF* formulae, and $\Gamma_{2,1}$ is nothing but the set of formulae in *CNF*. Now for any $t \geq 2$, we define the following problem.

Definition 45. #$\Gamma_{t,1}$-WSAT*: For any $t \geq 2$ consider the counting problem called* #$\Gamma_{t,1}$-WSAT*:*

***Input**: A formula ϕ over n variables of size m where $\phi \in \Gamma_{t,1}$ and an integer k ($\leq$ n).*

***Parameter**: $k.log(n)$*

***Task**: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be True ?*

Now we are going to define #*WK*-hierarchy that is basically the counting counterpart of the *WK*-hierarchy which is defined in [14].

Definition 46. *For $t \geq 2$, #$WK[t]$ is the class of counting problems that are parametric w-T-reducible to* #$\Gamma_{t,1}$-WSAT.

Similar to the decision version, it can also be proved that,

Theorem 23. *For $t \geq 2$, #$VC_t \subseteq$ #$WK[t]$.*

The original proof (for decision versions as mentioned in chapter 3, Theorem 20, using the results in [14]) is strongly parsimonious and work for counting version as well. Hence the above corollary directly follows from that. Similarly, it is also mentioned in chapter 3 that $WK[t] \subseteq VC_{t+1}$ (Theorem 20). But for counting versions, we prove the analogous result in similar but much simpler way. Corresponding reduction is going to be weakly parsimonious.

Theorem 24. *For $t \geq 2$, #$WK[t] \subseteq$ #VC_{t+1}.*

Proof. Let us consider a $\Gamma_{t,1}$ ($t \geq 2$) formula ϕ over n input variables. This ϕ is our $\#\Gamma_{t,1}$-WSAT instance which asks how many satisfying assignments of weight k are there for ϕ. To prove this theorem we are going to reduce this instance to a $\#VC_{t+1}$ instance ψ, which will be a circuit of depth at most $t+1$.

To prove this, we consider a problem corresponding to any general class of formula Φ. Given a class of Boolean formula Φ, we consider Φ^+, $\Phi^- \subseteq \Phi$ to denote the restrictions of Φ to formulas containing only positive and negative literals, respectively. Now we define the following problem:

#Φ-WSAT:

Input: A formula ϕ over n variables of size m where $\phi \in \Phi$.

Parameter: $k.log(n)$

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True* ?

Now we start the proof by dividing it into in two parts. Firstly we consider the case when t is even. We will consider the case of odd t later.

t is even:

For even t, Flum and Grohe has already shown a reduction (which is strongly w-parsimonious) ([25], Lemma 7.6 (1)) from $\#\Gamma_{t,1}$-WSAT to $\#\Gamma^+_{t,1}$-WSAT. Using the same reduction, we can reduce ϕ to another formula $\phi^{'}$ which contains only positive literals. ϕ has a satisfying assignment such that exactly k variables are assigned to be *True* iff $\phi^{'}$ has a satisfying assignment such that exactly $k^{'}$ variables are assigned to be *True*. Suppose number of variables of $\phi^{'}$ is $n^{'}$. $k^{'}$ and $n^{'}$ are polynomially bounded in k and n respectively. The reduction is strongly w-parsimonious.

Now we are going to reduce this $\#\Gamma^+_{t,1}$-WSAT instance $\phi^{'}$ to a $\#VC_{t+1}$ instance as follows. Suppose the input variables of $\phi^{'}$ are $x_0, x_1, \ldots, x_{n^{'}-1}$. Parameter for $\phi^{'}$ is $k^{'}.log(n^{'})$. We consider $l = \left\lceil log(n^{'}) \right\rceil$. Now we introduce following $k^{'}l$ variables, $X_{i,j}$, $1 \leq i \leq k^{'}$, $0 \leq j \leq l-1$. We use these variables with following interpretation. If the i^{th}-variable ($1 \leq i \leq k^{'}$) set to be *True* in $\phi^{'}$ is x_p ($0 \leq p \leq n^{'}-1$), $X_{i,0}X_{i,1}\ldots X_{i,l-1}$ will represent the binary string which is actually the binary encoding of p.

Using these newly introduced variables, we are going to construct a new Boolean formula. As t is even, the bottom level Boolean operation is $\vee$ for $\phi^{'}$. All the literals in the bottom level are positive literals. We now replace any variable x_i ($0 \leq i \leq n^{'}-1$) in $\phi^{'}$ by $\bigvee_{i=1}^{k^{'}}((X_{i,0} \oplus b_0) \wedge (X_{i,1} \oplus b_1) \wedge \ldots \wedge (X_{i,l-1} \oplus b_{l-1}))$ where $b_0 b_1 \ldots b_{l-1}$ is the binary representation of integer i, $(X_{i,j} \oplus b_j) = X_{i,j}$ if $b_j = 1$ and $(X_{i,j} \oplus b_j) = \neg X_{i,j}$ if $b_j = 0$, $0 \leq j \leq l-1$. After these replacements, we name the new formula by $\phi^{''}$. It is

easy to see that ϕ'' is a Boolean formula of depth $t+1$.

We can also see that if ϕ' has a satisfying assignment of weight k', $X_{i,j}$ variables of ϕ'' can correctly encode the k' variables of ϕ' assigned to be *True*. But in the opposite direction, it is not true yet. For that we have to ensure that all the k' variables of ϕ', encoded by $X_{i,j}$ variables of ϕ'' are different. So we construct another formula ψ' as follows.

$\psi' = \phi'' \wedge \bigwedge_{1 \leq i < j \leq k'} (\bigvee_{p=0}^{l-1} ((X_{i,p} \wedge \neg X_{j,p}) \vee (\neg X_{i,p} \wedge X_{j,p})))$.

If n' is not a power of 2, $X_{i,j}$ variables for ψ' can still choose some x_p where $p \geq n'$ but $p \leq 2^l - 1$. So we have to make sure that no such x_p is selected. It is easy to do. Suppose the binary representation of one such p ($\geq n'$ but $\leq 2^l - 1$) is $c_0 c_1 \ldots c_{l-1}$. So we will add following sub-formula by conjunction with ψ'.

$\bigwedge_{i=1}^{k'} ((X_{i,0} \oplus c_0) \vee (X_{i,1} \oplus c_1) \vee \ldots \vee (X_{i,l-1} \oplus c_{l-1}))$, where $(X_{i,j} \oplus c_j) = X_{i,j}$ if $c_j = 0$ and $(X_{i,j} \oplus c_j) = \neg X_{i,j}$ if $c_j = 1$, $0 \leq j \leq l-1$.

After taking care of all such unwanted variables, suppose the final formula is ψ. For $t \geq 2$, it is easy to see that ψ is a depth $t+1$ formula. This ψ is our final $\#VC_{t+1}$ instance. We can also see that for each satisfying assignment of weight k' for ϕ', ψ has $k'!$ satisfying assignments. So total number of satisfying assignments of weight k' for ϕ' is total number of satisfying assignments of ψ, divided by $k'!$. Hence clearly this reduction is weakly w-parsimonious and $\#WK[t] \subseteq \#VC_{t+1}$ when t is even.

Now we consider the other case.

t is odd:

For odd $t \geq 3$, we will reduce $\#\Gamma_{t,1}$-WSAT instance ϕ to a $\#\Delta_{t,1}$-WSAT instance ϕ_d by a very simple reduction. We will just take $\phi_d = \bar{\phi}$. Applying De Morgan's law, we can see that ϕ_d is a $\Delta_{t,1}$ formula and # of satisfying assignments of weight k for ϕ = $\binom{n}{k}$ - # of satisfying assignments of weight k for ϕ_d. Clearly this reduction is weakly w-parsimonious.

For odd $t \geq 3$, Flum and Grohe has already shown a reduction (which is strongly w-parsimonious) ([25], Lemma 7.6 (1)) from $\#\Delta_{t,1}$-WSAT to $\#\Delta^+_{t,1}$-WSAT. Using the same reduction, we can reduce ϕ_d to another formula ϕ'_d which contains only positive literals. ϕ_d has a satisfying assignment such that exactly k variables are assigned to be *True* iff ϕ'_d has a satisfying assignment such that exactly k' variables are assigned to be *True*. Suppose number of variables of ϕ'_d is n'. k' and n' are polynomially bounded in k and n respectively. The reduction is strongly w-parsimonious.

Now we will reduce $\#\Delta^+_{t,1}$-WSAT instance ϕ'_d to a $\#\Gamma^-_{t,1}$-WSAT instance ϕ' using the similar technique used just a while ago. We will just take $\phi' = \bar{\phi'_d}$. Applying De

Morgan's law, we can see that $\phi^{'}$ is a $\Gamma^{-}_{t,1}$ formula and # of satisfying assignments of weight $k^{'}$ for $\phi^{'}_{d} = \binom{n^{'}}{k^{'}}$ - # of satisfying assignments of weight $k^{'}$ for $\phi^{'}$. Clearly this reduction is weakly w-parsimonious.

Now we are going to reduce this $\#\Gamma^{-}_{t,1}$-WSAT instance $\phi^{'}$ to a $\#VC_{t+1}$ instance as follows. Suppose the input variables of $\phi^{'}$ are $x_0, x_1, \ldots, x_{n^{'}-1}$. Parameter for $\phi^{'}$ is $k^{'}.log(n^{'})$. We consider $l = \left\lceil log(n^{'}) \right\rceil$. Now we introduce following $k^{'}l$ variables, $X_{i,j}$, $1 \leq i \leq k^{'}$, $0 \leq j \leq l-1$. We use these variables with following interpretation. If the i^{th}-variable ($1 \leq i \leq k^{'}$) set to be *True* in $\phi^{'}$ is x_p ($0 \leq p \leq n^{'}-1$), $X_{i,0}X_{i,1}\ldots X_{i,l-1}$ will represent the binary string which is actually the binary encoding of p.

Using these newly introduced variables, we are going to construct a new Boolean formula. As t is odd, the bottom level Boolean operation is $\wedge$ for $\phi^{'}$. All the literals in the bottom level are negative literals. We now replace any literal $\bar{x_i}$ ($0 \leq i \leq n^{'}-1$) in $\phi^{'}$ by $\bigwedge_{i=1}^{k^{'}}((X_{i,0} \oplus b_0) \vee (X_{i,1} \oplus b_1) \vee \ldots \vee (X_{i,l-1} \oplus b_{l-1}))$ where $b_0 b_1 \ldots b_{l-1}$ is the binary representation of integer i, $(X_{i,j} \oplus b_j) = X_{i,j}$ if $b_j = 0$ and $(X_{i,j} \oplus b_j) = \neg X_{i,j}$ if $b_j = 1$, $0 \leq j \leq l-1$. After these replacements, we name the new formula by $\phi^{''}$. It is easy to see that $\phi^{''}$ is a Boolean formula of depth $t+1$.

Remaining part of the proof is same as the case when t is even. Because, we can also see that if $\phi^{'}$ has a satisfying assignment of weight $k^{'}$, $X_{i,j}$ variables of $\phi^{''}$ can correctly encode the $k^{'}$ variables of $\phi^{'}$ assigned to be *True*. But in the opposite direction, it is not true yet. For that we have to ensure that all the $k^{'}$ variables of $\phi^{'}$, encoded by $X_{i,j}$ variables of $\phi^{''}$ are different. So we construct another formula $\psi^{'}$ as follows.

$\psi^{'} = \phi^{''} \wedge \bigwedge_{1 \leq i < j \leq k^{'}}(\bigvee_{p=0}^{l-1}((X_{i,p} \wedge \neg X_{j,p}) \vee (\neg X_{i,p} \wedge X_{j,p})))$.

If $n^{'}$ is not a power of 2, $X_{i,j}$ variables for $\psi^{'}$ can still choose some x_p where $p \geq n^{'}$ but $p \leq 2^l - 1$. So we have to make sure that no such x_p is selected. It is easy to do. Suppose the binary representation of one such p ($\geq n^{'}$ but $\leq 2^l - 1$) is $c_0 c_1 \ldots c_{l-1}$. So we will add following sub-formula by conjunction with $\psi^{'}$.

$\bigwedge_{i=1}^{k^{'}}((X_{i,0} \oplus c_0) \vee (X_{i,1} \oplus c_1) \vee \ldots \vee (X_{i,l-1} \oplus c_{l-1}))$, where $(X_{i,j} \oplus c_j) = X_{i,j}$ if $c_j = 0$ and $(X_{i,j} \oplus c_j) = \neg X_{i,j}$ if $c_j = 1$, $0 \leq j \leq l-1$.

After taking care of all such unwanted variables, suppose the final formula is ψ. For odd $t \geq 3$, it is easy to see that ψ is a depth $t+1$ formula. This ψ is our final $\#VC_{t+1}$ instance. We can also see that for each satisfying assignment of weight $k^{'}$ for $\phi^{'}$, ψ has $k^{'}!$ satisfying assignments. So total number of satisfying assignments of weight $k^{'}$ for $\phi^{'}$ is total number of satisfying assignments of ψ, divided by $k^{'}!$. Hence clearly this reduction is weakly w-parsimonious and $\#WK[t] \subseteq \#VC_{t+1}$ for both odd and even $t \geq$

2. □

All the parametric decision problems in complexity class P, are trivially in VC_0. But corresponding counting versions may not be in $\#VC_0$. We will see a few such examples later. If the parameter of any parametric problem is polynomially bounded in the problem input size, once again the problem is trivially in VC_0 (e.g. 3SAT). For those problems, corresponding counting versions are in $\#VC_0$ as well. Now we are going to see a problem, for which both the decision and counting versions are compressible for non-trivial reasons.

VERTEX COVER:

Input: A graph $G(V,E)$ with n vertices, and an integer k.

Task: Is there a vertex cover (a set $D \subseteq V$, such that for every edge $\{u,v\} \in E$, either u or v are in D) of size k in G?

Parameter: $k.log(n)$

Corresponding counting version is,

#VERTEX COVER:

Input: A graph $G(V,E)$ with n vertices, and an integer k.

Task: How many vertex covers of size k are there in G?

Parameter: $k.log(n)$

It is already proved in [13] (Claim 2.1) that VERTEX COVER is in VC_0. We are now going to use the same compression algorithm to show that it works for counting version as well to prove the next corollary. The original algorithm (standard polynomial kernelization) can be seen in [44] using the technique presented in [21].

Corollary 5. #VERTEX COVER $\in \#VC_0$.

Proof. Suppose, a graph $G(V,E)$ with n vertices is given to us. Now, if a vertex cover D of size k exists in G, any vertex of degree greater than k must be inside the set D. The compression algorithm will simply identify all such vertices and list them in the cover, while removing all their outgoing edges (because they do not need to be covered by other vertices). Suppose the algorithm has identified $k^{'} \leq k$ vertices like that. The graph left after this process (let us name it by $G^{'}$) has degree at most k, and furthermore all edges have at least one end in the cover. Now it is clear that G has a vertex cover of size k iff $G^{'}$ has a vertex cover of size $k-k^{'}$. Thus, if the original graph G has a vertex cover of size k, the total number of edges left in $G^{'}$ is at most k^2 (at most k vertices in the vertex cover for $G^{'}$ where each of them covering at most k edges each). If there are more then k^2 edges then the answer to the problem is *NO*, otherwise, such a graph

can be represented by listing all edges, which takes $O(k^2.log(k))$ bits. If the answer to the problem is *NO*, the compression algorithm will output a trivial compressed graph with no vertex cover of size $k - k^{'}$. It is now easy to see that the total number of vertex covers of the original #VERTEX COVER instance is same as the total number of vertex covers of the reduced #VERTEX COVER instance. Hence, #VERTEX COVER is Weakly-Parsimonious compressible and present in $\#VC_0$. □

4.2 Completeness Results

We are now going to consider some interesting counting problems and going to show the hardness and completeness results corresponding to that.

#SAT:

Input: A formula ϕ in *CNF* with n variables.

Task: How many satisfying assignments are there for ϕ ?

Parameter: n

#NAE-SAT:

Input: A formula ϕ in *CNF* with n variables.

Task: How many satisfying assignments are there for ϕ such that each clause of ϕ contains at least one *True* and one *False* literal ?

Parameter: n

We have already shown that the decision problem NAE-SAT is VC_2-complete (chapter 3, Theorem 12). Now we prove the following.

Theorem 25. #NAE-SAT *is* $\#VC_2$*-complete.*

Proof. Firstly we will show that #NAE-SAT $\in \#VC_2$. Let us consider an NAE-SAT instance ϕ. Now for each clause c_i we will construct another clause $c_i^{'}$ in the following way: for each literal $l \in c_i$ we put $\neg l$ in $c_i^{'}$. Now inserting all the new clauses into ϕ we will construct a new formula ψ. So ψ is twice in size with respect to ϕ. Both of them have the same variables. We can easily see that ψ is satisfiable iff ϕ is *NAE*-satisfiable. So clearly this is a strongly w-parsimonious reduction and hence #NAE-SAT $\in \#VC_2$.

We are now going to show a weakly w-parsimonious reduction in the opposite direction (we have used this reduction before in chapter 3, Theorem 13).

Let us now consider a SAT instance ϕ over $x_1, x_2, \ldots, x_n$. Let us consider $c_1, c_2, \ldots, c_l$ are the clauses of ϕ. Now we consider two separate variables y and z, other than $x_1, x_2, \ldots, x_n$. We modify each clause c_i as $c_i^{'} = c_i \vee \neg y$ and introduce a new clause $c^{'}$

$= y \vee z$. Now our modified formula $\phi' = c_1' \wedge c_2' \wedge \ldots \wedge c_l' \wedge c'$. Now we are going to prove that ϕ is satisfiable iff ϕ' is *NAE*-satisfiable.

Suppose ϕ is satisfiable. Now for any satisfying assignments to $x_1, x_2, \ldots, x_n$ for ϕ, we will use the same assignments for ϕ'. Now we will assign $y = 1$ and $z = 0$. We can now see that ϕ' is *NAE*-satisfiable. So for any truth assignments for ϕ there exists a truth assignments for ϕ' satisfying *NAE-SAT* property.

Now suppose ϕ' is *NAE*-satisfiable. So obviously either $y = 1$ and $z = 0$ or $y = 0$ and $z = 1$. If $y = 1$, we will use the same assignments of $x_1, x_2, \ldots, x_n$ from ϕ' to ϕ. If $y = 0$, we will invert the assignments of all the variables in ϕ'. We can understand that still it's *NAE*-satisfiable. Now we will use the same procedure to assign $x_1, x_2, \ldots, x_n$ for ϕ. Clearly $\phi \in SAT$.

Here we can see that the parameter, the number of variables, is increased by just two. So the above reduction is *w*-reduction. Now we are going to prove that this reduction is parsimonious. For that we have to calculate the total number of satisfying assignments of ϕ from total number of *NAE*-satisfying assignments of ϕ'.

Now let us consider that N is the total number of satisfying assignments of ϕ. We also consider that N_1 is the total number of *NAE* satisfying assignments for ϕ. It means each of those N_1 assignments will keep at least one true and one false literal in each of the clauses in ϕ. N_2 is the other type satisfying assignments for ϕ. So clearly $N = N_1 + N_2$. Now we will try to calculate the total number of *NAE* satisfying assignments for ϕ'. Now for each of the N_1 assignments, we can set either $y = 1$ and $z = 0$ or $y = 0$ and $z = 1$ for ϕ'. Either of those will make ϕ' *NAE* satisfiable. So total number of witnesses for ϕ', corresponding N_1 will be $2.N_1$. For each of the N_2 assignments, we have to set $y = 1$ and $z = 0$ for ϕ' to make it *NAE* satisfiable. So total number of witnesses, corresponding N_2 in ϕ will be again N_2 for ϕ'. But we can observe, if we flip the truth assignments of all these N_2 witnesses corresponding to $(n + 2)$ variables in ϕ', it will still be the *NAE* satisfying assignments for ϕ'. In this way we can get another set of witnesses for ϕ' which we haven't considered before. So total number of *NAE* satisfying assignments for ϕ' is $N' = 2(N_1 + N_2)$. So we can easily calculate the total number of satisfying assignments of ϕ from total number of *NAE*-satisfying assignments of ϕ' by dividing it by 2. So clearly the reduction is weakly parametric *w*-parsimonious. Hence #NAE-SAT is $\#VC_2$-complete. □

Let us now consider the following counting problem.

#SET SPLITTING:

Input: A set S of n elements and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S.
Task: How many ways the set S can be partitioned into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the sets in $\mathcal{C}$ is contained in either of s_1 and s_2 ?
Parameter: n

In chapter 3, we have already shown that SET SPLITTING is VC_2-complete (Theorem 11). There, it is also shown that SET SPLITTING and NAE-SAT are w-reducible to each other (reduction in Theorem 15 and 11 ; although the reduction in Theorem 15 is shown for SET SPLITTING$_k$, as mentioned there, the same reduction works for SET SPLITTING as well). Using the same reductions, we can also prove that,

Theorem 26. #SET SPLITTING *is* #VC_2*-complete.*

As shown in chapter 3, the reduction from SET SPLITTING to NAE-SAT (first part of reduction in the proof Theorem 11) is eventually strongly parsimonious and works for corresponding counting problems as well. For the reduction in the opposite direction (reduction in Theorem 15), it is also easy to see that for any set splitting for the SET SPLITTING problem instance $(S, \mathcal{C})$, the reduced NAE-SAT instance, formula ϕ, has exactly 2 satisfying assignments such that each clause of ϕ contains at least one *True* and one *False* literal. So clearly, the reduction is weakly w-parsimonious and hence, the above theorem directly follows from it.

Let us now define one very simple problem #DNF.

#DNF:
Input: A formula ϕ in disjunctive normal form of size m over n variables.
Task: How many satisfying assignments are there for ϕ ?
Parameter: n

It is easy to see that the decision version of this problem is easily solvable in polynomial time (as we can make any particular clause satisfiable just making all its literals *True*) and hence in VC_0. But for counting version, we prove the following result.

Theorem 27. #DNF *is* #VC_2*-complete.*

Proof. We will show weakly parametric w-parsimonious reduction from #DNF to #SAT and vice versa to prove this result. Let us consider a formula ϕ in disjunctive normal form of size m over n variables. This ϕ is our #DNF instance where we ask how many satisfying assignments are there for ϕ. We take another formula $\psi = \bar{\phi}$. Applying De Morgan's law, we can see that ψ is in conjunctive normal form and hence our SAT instance for this reduction. Clearly this reduction from ϕ to ψ is in

polynomial time in m. If the total number of satisfying assignments for ϕ and ψ are N_1 and N_2 respectively, we can observe that $N_1 = 2^n$ - N_2. So clearly the reduction is weakly parametric w-parsimonious. Hence #DNF $\in \#VC_2$.

In the similar manner, taking complement of the SAT instance, we can find a weakly parametric w-parsimonious reduction from #SAT to #DNF. Hence #DNF is $\#VC_2$-complete. □

Now we define the weighted version of the previous problem and prove the completeness result mentioned below.

#WEIGHTED-DNFSAT:

Input: A formula ϕ in disjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

Before proving the next result we define the following problem.

#WEIGHTED-CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

From the definition (Definition 46) it is easy to understand that #WEIGHTED-CNFSAT is $\#WK[2]$-complete.

Theorem 28. #WEIGHTED-DNFSAT *is* $\#WK[2]$*-complete.*

Proof. The proof is very much similar to the previous one. We will show weakly parametric w-parsimonious reduction from #WEIGHTED-DNFSAT to #WEIGHTED-CNFSAT and vice versa to prove this result. Let us consider a formula ϕ in disjunctive normal form of size m over n variables. This ϕ is our #WEIGHTED-DNFSAT instance where we ask how many satisfying assignments are there for ϕ, such that exactly k variables are assigned to be *True*. We take another formula $\psi = \bar{\phi}$. Applying De Morgan's law, we can see that ψ is in conjunctive normal form and hence our reduced #WEIGHTED-CNFSAT instance. Clearly this reduction from ϕ to ψ is in polynomial time in m. If the total number of satisfying assignments of weight k for ϕ and ψ are N_1 and N_2 respectively, we can observe that $N_1 = \binom{n}{k}$ - N_2. So clearly the reduction is weakly parametric w-parsimonious. Hence #WEIGHTED-DNFSAT $\in \#WK[2]$.

In the similar manner, taking complement of the #WEIGHTED-CNFSAT instance, we can find a weakly parametric w-parsimonious reduction from the counting problem #WEIGHTED-CNFSAT to #WEIGHTED-DNFSAT. Hence #WEIGHTED-DNFSAT is $\#W[2]$-complete. □

The decision version of #WEIGHTED-DNFSAT is easily solvable in polynomial time and hence in VC_0. But for counting version, we have proved that it is $\#W[2]$-complete. Below, we have briefly described how to prove that the decision problem WEIGHTED-DNFSAT is solvable in polynomial time.

Let us consider a formula ϕ in disjunctive normal form of size m over n variables (if there is any clause containing both positive and negative literals corresponding any single variable, we can easily remove them in polynomial time replacing it by 0.). Suppose another integer k ($\leq n$) is given. Now, we will search for a clause in ϕ which contains $\leq (n-k)$ negative literals and $\leq k$ positive literals.

Suppose we have found one such clause which contains k_1 ($\leq (n-k)$) negative literals and k_2 ($\leq k$) positive literals. Now we will assign all the variables corresponding to negative literals in that clause to 0 and all the variables corresponding to positive literals to 1. So k_2 variables are so far assigned to be *True*. Now we are sure that there are at least $(k-k_2)$ variables left un-assigned (as $k_1 \leq (n-k)$). So we will pick any $(k-k_2)$ variables from the un-assigned variables arbitrarily and assign them 1. We will assign any remaining un-assigned variables to 0. So we have found one satisfying assignment for ϕ such that exactly k variables are assigned to be *True*.

Now suppose there is no such clause present in ϕ. So if we try to make ϕ satisfiable, we have to set more than $(n-k)$ variables to 0 (i.e., less than k variables set to 1) or more than k variables set to 1. So we can say that there is no satisfying assignment for ϕ such that exactly k variables are assigned to be *True*. Hence, WEIGHTED-DNFSAT $\in VC_0$.

Let us now consider the following definition.

#CLIQUE:

Input: A graph G with n vertices, and an integer k.

Task: How many cliques of size k (a pairwise adjacent subset of k vertices) are there in G ?

Parameter: $k.log(n)$

In chapter 3, we have already mentioned the proof of VC_1-completeness of the CLIQUE problem (Theorem 9). It is easy to see that same reductions work for counting

problems as well and hence, we can conclude that,

Proposition 27. #CLIQUE *is* #VC_1*-complete.*

In chapter 3, section 3.2.2, we have shown several reductions related to VC_1 problems. It is easy to see that all of them work for corresponding counting problems as well. As a result we get more harness and completeness results corresponding to class #VC_1. Although most of them are strongly w-parsimonious, all of them are not. Below we show a reduction which is actually weakly w-parsimonious. For that we define the counting version of MULTICOLOURED CLIQUE problem that we have considered before (Proposition 13).

#MULTICOLOURED CLIQUE:

Input: A graph $G = (V, E)$ with $|V| = n$, an integer k and a colouring function $c : V \rightarrow [k]$.

Task: How many multicoloured cliques of size k (a clique containing exactly one vertex of each colour) are there in G ?

Parameter: $k.log(n)$

Proposition 28. #CLIQUE *is weakly w-parsimonious reducible to* #MULTICOLOURED CLIQUE.

Proof. Let us consider that a graph $G(V,E)$ with n vertices are given to us with another integer k ($k \leq n$). From this graph G we construct another graph $G^{'}$ as follows. For each vertex $v_i \in V$, $1 \leq i \leq n$, we construct k copies of the same vertices, $v_{i,1}, v_{i,2}, \ldots, v_{i,k}$, colouring each vertex $v_{i,j}$ with colour j, $1 \leq j \leq k$. We add two vertices $v_{p,q}$ and $v_{r,s}$ ($q \neq s$) in $G^{'}$ by an edge iff v_p and v_r are connected by an edge in G. It is now easy to see that G has a clique of size k iff $G^{'}$ has a multicoloured clique of size k. We can also see that for any clique of size k in G, there will be exactly $k!$ multicoloured cliques of size k in $G^{'}$, one corresponding to each of $k!$ permutations of k colours. Hence, it is clearly a weakly w-parsimonious reduction and the result is proved. □

Now we will consider slightly different kind of problems. Suppose $G(V,E)$ is a graph with n vertices and m edges. We will start with the definition of a property in a graph.

Definition 47. ***Complete-CycleSet:*** *Complete-CycleSet is a set of vertex disjoint cycles in a graph, removal of whose edges makes the graph cycle free.*

We want to point out here that the problem of finding a *Complete-CycleSet* in a graph is in complexity class P. Using Depth-first search algorithm we can find a cycle and remove all the edges of that cycle from the graph. And then keep doing the same thing until we find that the graph is cycle free. Clearly we can do that in polynomial time as Depth-first search algorithm can be run in $O(m)$ time.

Definition 48. ***Unique-cycle:*** *Unique-cycle is a cycle in a graph containing at least one edge, participating in no other cycle except this.*

Definition 49. ***Unique complete-cycleSet:*** *Unique complete-cycleSet is a set of vertex disjoint unique-cycles in a graph, removal of whose edges makes the graph cycle free.*

It is easy to see that the problem of finding a *Unique complete-cycleSet* in a graph is also in complexity class P as checking whether a cycle is unique or not, can be done again by depth first search.

Now we are formally going to define parametric counting problem related to this and show that the problem is $\#VC_2$-hard.

Definition 50. #K-UNIQUE COMPLETE-CYCLESET*:*
Input*: A directed graph $G(V,E)$ with n vertices and m edges and an integer $K \leq m$.*
Parameter*: $K.log(m)$*
Task*: How many Unique complete-cycleSet are there in the graph containing at most K unique-cycles ?*

The proof of the hardness result of this problem is inspired by the technique used in [1]. There they have given an alternate proof of #P-completeness of Zero-One-Permanent. This result was originally proved by L. G. Valiant [33]. Before proving that result, we are going to give some more definitions.

Definition 51. *We say that a n-node weighted directed graph $G(V,E)$ and a $n \times n$ matrix A correspond to one another if for every i, $j \in \{1,2,\ldots,n\}$, $A_{i,j}$ is the weight of the edge $i \rightarrow j$.*

Definition 52. *Suppose $R \subseteq E$ is a K-Unique complete-cycleSet. The weight of R, denoted by $W(R)$, is the product of the weights of the edges in R.*

Now, we are ready to prove the result.

Theorem 29. #K-UNIQUE COMPLETE-CYCLESET *is $\#VC_2$-hard.*

Proof. We will prove this by finding a weakly parsimonious reduction from #SAT to #K-UNIQUE COMPLETE-CYCLESET. Suppose ϕ is a SAT instance with n variables. We will firstly construct a weighted directed graph G_ϕ corresponding to ϕ such that there is a mapping between assignments for ϕ and K-Unique complete-cycleSet in G_ϕ. We will take $K = n$. This mapping satisfies the following conditions:

- The sum of weights of all the Unique complete-cycleSets which correspond to each satisfying assignment is equal to 1.
- The sum of weights of all the other Unique complete-cycleSet is equal to 0.

Construction of G_ϕ using Clause Components:
The graph G_ϕ is constructed as follows:

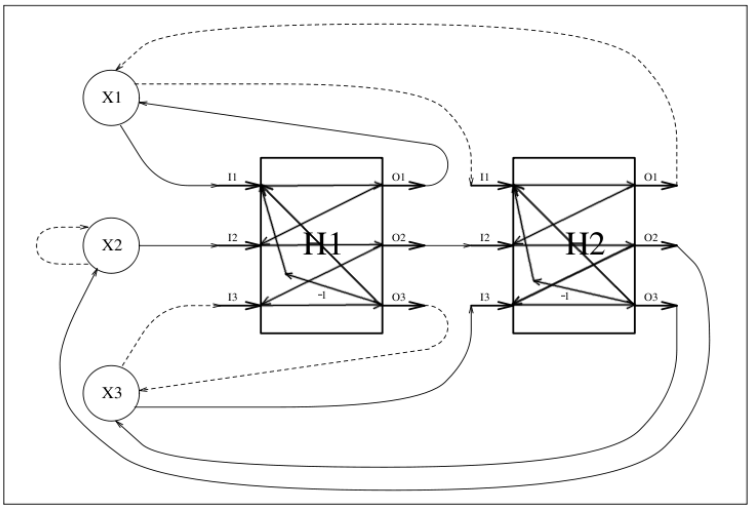
- For each variable x_i in ϕ there is a node in G. We refer to these nodes as *variable nodes*.
- For each clause $c_j = (\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_{t_j})$ in ϕ, there is a clause component in G_ϕ, denoted by H_j. The literals in the clauses are sorted with respect to their indices. The clause component has t_j input edges labelled I_1, I_2, …, I_{t_j}, and t_j output edges labelled O_1, O_2, …, O_{t_j}. These edges connect the component to other components or to variable nodes. Intuitively, the edges I_t and O_t $(1 \leq t \leq t_j)$ of H_j correspond to the literal α_t in c_j.
- For each variable x_i in ϕ we form two *cycles* in the graph G_ϕ.
  - Let $c_{j_1}, c_{j_2}, \ldots, c_{j_l}$ be the clauses that contain the literal x_i in the order they appear in ϕ. The T-cycle of x_i starts at the variable node x_i, visits the clause components $H_{j_1}, H_{j_2}, \ldots, H_{j_l}$ and goes back to x_i. If x_i is the k_{th} literal in a clause c_j, then the T-cycle of x_i enters H_j through the input edge I_k and exits it through the output edge O_k.
  - Let $c_{j_1'}, c_{j_2'}, \ldots, c_{j_l'}$ be the clauses that contain the literal $\bar{x_i}$ in the order they appear in ϕ. The F-cycle of x_i starts at the variable node x_i, visits the clause components $H_{j_1'}, H_{j_2'}, \ldots, H_{j_l'}$ in the similar way and goes back to x_i.

  Formally, there is an edge from the output edge O_{k_1} of H_{j_1} to the input edge I_{k_2} of H_{j_2} if the next occurrence of the k_1^{th} literal of the clause c_{j_1} is as the k_2^{th} literal of the clause c_{j_2}. If the literal x_i does not appear in ϕ, then the T-cycle of x_i is a

self loop, and the same goes for $\bar{x_i}$. The weights of all the edges in the T-cycles and F-cycles of every literal are one.

- If $c_j = (\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_{t_j})$ we denote input and output nodes as i_1, i_2, …, i_{t_j} and o_1, o_2, …, o_{t_j} inside the clause component H_j. We will add edges as follows: $\{i_1, o_1\}$, $\{o_1, i_2\}$, $\{i_2, o_2\}$, $\{o_2, i_3\}$, …, $\{o_{t_{j-1}}, i_{t_j}\}$, $\{i_{t_j}, o_{t_j}\}$, $\{o_{t_j}, i_1\}$. All these edges are of weight 1. We will introduce another extra node inside each clause component, say p_j. We will add edges $\{o_{t_j}, p_j\}$ and $\{p_j, i_1\}$ as well. $\{o_{t_j}, p_j\}$ is of weight -1 and $\{p_j, i_1\}$ is of 1.

For the formula $\phi = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2 \vee x_3)$, corresponding graph G_ϕ is shown below.



In the diagram, we can see that, inside each clause component, there are 2 unique cycles, one of weight 1, and another of weight -1 (taking either of the two paths connecting output node of the last literal and input node of the first literal). Let us denote T-cycle and F-cycle as external cycles as they are containing edges outside clause components. Now we can see that, in any *Unique complete-cycleSet*, if there is a T-cycle or F-cycle through a clause component, say H_j, none of those internal cycles can not be there in the *Unique complete-cycleSet*. Because in that case, the cycles won't be disjoint. Now corresponding to any assignment for n variables in ϕ, we will uniquely map a set of external cycles as follows:

- If variable x_i is assigned to 1, we will take T-cycle corresponding to vertex x_i in G_ϕ.
- If variable x_i is assigned to 0, we will take F-cycle corresponding to vertex x_i in G_ϕ.

It is easy to see that, for any assignment in ϕ there exists at least one *Unique complete-cycleSet* in G_ϕ. For any assignment, if certain clause component is not touched by corresponding T-cycle or F-cycle, we can choose either of the internal cycle to make it a *Unique complete-cycleSet*.

Now we observe that,

- Any satisfying assignment for the formula ϕ will correspond to exactly one *K-Unique complete-cycleSet* where $K = n$. Weight of that *K-Unique complete-cycleSet* will be 1.
- For any unsatisfying assignment, sum of weights of all the *K-Unique complete-cycleSet* will be 0. There can be *K-Unique complete-cycleSet* which does not correspond to any assignment (taking input and output edges of different literals). Sum of weights of all such *K-Unique complete-cycleSet* will be 0 as well.

Above statements are easy to see as internal cycles are constructed in pairs, one of weight 1, another of weight -1. So in any *K-Unique complete-cycleSet*, we can choose either of them or none of them. In all the cases, sum of weights of all the *K-Unique complete-cycleSet* will be 0. Hence,

Total number of satisfying assignments in ϕ = Sum weights of all the *K-Unique complete-cycleSet* in G_ϕ.

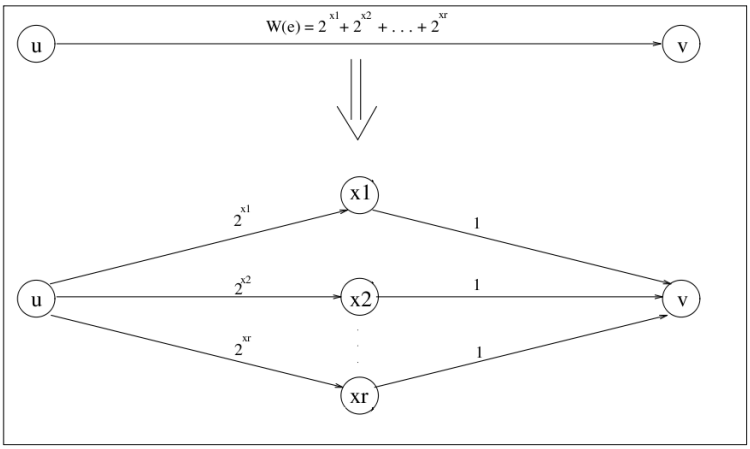
We can see that the graph G_ϕ is actually a weighted graph. Now we will convert the graph G_ϕ into an unweighted directed graph, say $G^{'}$ such that the sum of weights of all the *K-Unique complete-cycleSet*s becomes same as the number of *K-Unique complete-cycleSet*s in the graph.

Conversion from negative weight to positive weight:
For doing that we will use the similar technique used in [1], Section 5.1. Let us consider that A is the adjacency matrix of G_ϕ. We also consider that $Sum(A)$ = Sum weights of all the *K-Unique complete-cycleSet*s in the graph corresponding to A. We will obtain another $n \times n$ matrix from A as follows:

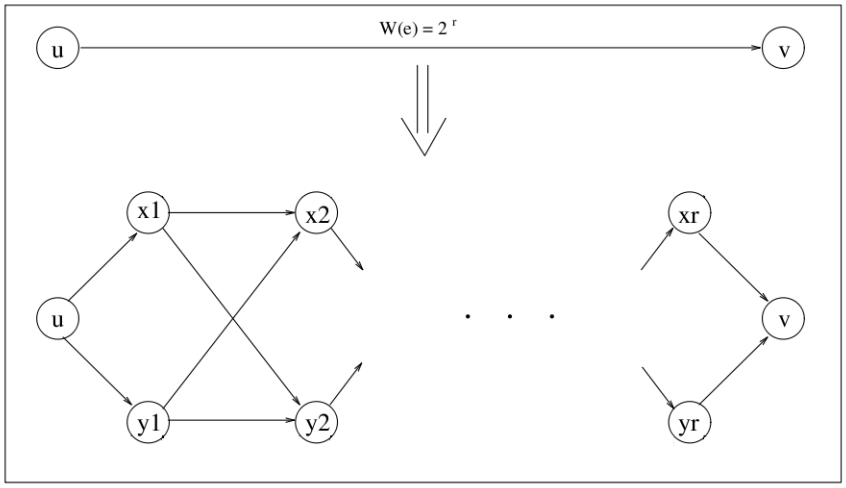
- We know that $Sum(A) < 2^{n^2}$. So we will choose $Q = 2^{n^2+1} + 1$.
- Then we will compute $A^{'} = A \ mod \ Q$.
- After that we compute $P = Sum(A^{'}) \ mod \ Q$.
- if $P < Q/2$ then $Sum(A) = P$. Otherwise $Sum(A) = Q$ - P.

We can observe that the transformation from A into $A^{'}$ is polynomial in n as the number of bits that we need to write Q is polynomial in n. The graph corresponding to $A^{'}$ doesn't contain any edge of negative weight.

Conversion from any positive weights to weights of 0 **or powers of** 2**:**
For doing that we will use the same technique used in [1], Section 5.2. It is shown in the diagram below. Basically, we will replace any edge $\{u,v\}$ by a sub-graph as shown in the diagram. It is now easy to see that, if the edge $\{u,v\}$ is taken in any *Unique complete-cycleSet*, sum of weights of all the paths from u to v is equal to the weight of the original edge $\{u,v\}$.



Conversion from weights of powers of 2 **to weight** 0 **or** 1**:**
For doing that we will use the same technique used in [1], Section 5.3. It is shown in the diagram below. Similarly, we will replace any edge $\{u,v\}$ by a sub-graph as shown in the diagram. It is now easy to see that, if the edge $\{u,v\}$ is taken in any *Unique complete-cycleSet*, sum of weights of all the paths from u to v is equal to the weight of the original edge $\{u,v\}$.



After using all the above reductions, we will finally get the required graph $G^{'}$. Total number of *K-Unique complete-cycleSet*s in the graph $G^{'}$ modulo Q is same as the total number of satisfying assignments of ϕ. Here the parameter of the #*K*-UNIQUE COMPLETE-CYCLESET problem is $K.log(m)$ where m is the number of edges in $G^{'}$. Here $K = n$ and clearly $m \leq 2^{poly(n)}$. Parameter of the #SAT problem is n. So clearly the reduction is keeping the parameter of the reduced problem polynomially bounded. Hence it is a parsimonious reduction and #*K*-UNIQUE COMPLETE-CYCLESET is $\#VC_2$-hard. □

Now we are going to define some problems which will be useful to prove an interesting theorems below. We have already considered some of them before.

SUBSET SUM:

Input: An integer k, a set S of m non-negative integers, and another integer B.

Task: Decide whether there are k integers in S which sum up to exactly B.

Parameter: $k.log(m)$

This decision problem is VC_1-hard (chapter 3, Theorem 9 and Figure 3.1).

#SUBSET SUM:

Input: An integer k, a set S of m non-negative integers, and another integer B.

Task: How many sub-sets of k integers of S are there, elements of which sum up to exactly B ?

Parameter: $k.log(m)$

As all the *w*-reductions for the decision problem are parsimonious and work for counting problems as well, we can say that #SUBSET SUM problem is $\#VC_1$-hard.

KNAPSACK:

Input: An integer k, a set T of m non-negative integers, and another integer A.

Task: Decide whether there are k integers in T which sum up to less than or equal to A.

Parameter: $k.log(m)$

This decision problem is in P. We can sort all the m integers in $O(m.log(m))$ time and take smallest k integers. If summation of all these smallest k integers are less than or equal to A then the answer to this problem is *yes*, other wise *no*.

#KNAPSACK:

Input: An integer k, a set T of m non-negative integers, and another integer A.

Task: How many sub-sets of k integers of T are there, elements of which sum up to less than or equal to A ?

Parameter: $k.log(m)$

Theorem 30. #KNAPSACK *is* #VC_1*-hard.*

Proof. We will prove this theorem by showing a w-T-reduction from #SUBSET SUM to #KNAPSACK. Let us consider the following #SUBSET SUM problem instance.

Suppose $S = \{a_1, a_2, \ldots, a_m\}$. *value* is a function which maps each a_i to some non-negative integer value, $value : S \rightarrow \mathbb{N}$. #SUBSET SUM problem finds, how many sub-sets of k integers of S are there, elements of which sum up to exactly B. If we consider F to denote the #SUBSET SUM counting problem, and x to be the problem instance described above, then $F(x, k.log(m)) = |\{S^{'} \mid S^{'} \subseteq S, |S^{'}| = k, \sum_{b_i \in S^{'}} value(b_i) = B\}|$.

Now we are going to construct a #KNAPSACK instance from the above instance as follows. $T = S = \{a_1, a_2, \ldots, a_m\}$. *value* is a function which maps each a_i to some non-negative integer value, $value : T \rightarrow \mathbb{N}$. #KNAPSACK problem finds, how many sub-sets of k integers of S are there, elements of which sum up to less than or equal to A. We will take $A = B$. If we consider G to denote the #KNAPSACK counting problem, and y to be the problem instance described above, then $G(y, k.log(m)) = |\{T^{'} \mid T^{'} \subseteq T, |T^{'}| = k, \sum_{b_i \in T^{'}} value(b_i) \leq A\}|$.

Now we have to prove that the reduction described above is actually parametric w-T-reduction. For that we are going to construct σ and τ. σ is reducing a F instance to a G instance. It is clear that $\sigma(x, k.log(m)) = (y, k.log(m))$ and σ is computable in time polynomial in m. Now we have to show that some polynomial time computable algorithm τ exists which is basically the algorithm with an oracle for G as in Definition 38). So, we are going to construct an algorithm (or function) τ which takes problem instance x as input, uses functions G and σ and computes $F(x, k.log(m))$ in

time polynomial in m.

The algorithm τ will work as follows. It will first construct another #SUBSET SUM problem instance $x^{'}$ from x. $x^{'}$ consists of same set $S = \{a_1, a_2, \ldots, a_m\}$. But this new problem finds, how many sub-sets of k integers of S are there, elements of which sum up to exactly (B-1). So instead of B, the sack size is (B-1), that is the only difference. We denote this slightly changed problem as $x^{'}$. Clearly the algorithm can construct this new problem in constant time. Now τ will compute $F(x, k.log(m))$ as follows, $F(x, k.log(m)) = G(\sigma(x, k.log(m)))$ - $G(\sigma(x^{'}, k.log(m)))$. Clearly this calculation is correct. We can also observe that the parameter is same throughout. Hence the reduction is parametric w-T-reduction and #KNAPSACK is $\#VC_1$-hard. □

We are now going to define another interesting problem.

MULTI CLIQUE-COLOURING:

Input: An undirected graph $G(V, E)$ over n vertices and l disjoint cliques (there is no edge $\{u, v\} \in E$ such that u and v are part of two different cliques) inside the graph, each of size k.

Task: Is the graph k colourable ?

Parameter: $(n - kl).log(k)$

Similarly we can define the counting version of this problem. But before that, we have to prove that $(n - kl).log(k)$ is a valid parameter for this problem. For that, we have to prove,

Proposition 29. *If all the neighbouring vertices of a clique of size k in a graph G is coloured by k colours, we can check in polynomial time (polynomial in size of graph G) if there is a valid k colouring for that clique.*

Proof. We prove this result by constructing a separate graph. Suppose the vertices in the clique are $V_1, V_2, \ldots, V_k$. We denote the k colours by $C_1, C_2, \ldots, C_k$. Let $V = \{V_1, V_2, \ldots, V_k\}$ and $C = \{C_1, C_2, \ldots, C_k\}$. Now for each vertex V_i, $1 \leq i \leq k$, we construct a list L_i which contains all the colours by which it can be coloured. Basically we will just check all the neighbouring vertices of V_i and their colours. Then we will construct the list L_i by removing those colours from the set C. Clearly we can construct the lists $L_1, L_2, \ldots, L_k$ in time polynomial in n, number of vertices in G. If any of those lists is empty, we will conclude that there is no valid k colouring for that clique. Otherwise we will construct a new graph $G^{'}$ as follows.

$G^{'}$ will contain 2 sets of vertices, one for each of V and C. For each $V_i \in V$, we will add vertex N_{V_i} and for each $C_i \in C$, we will add vertex N_{C_i} in $G^{'}$. For each colour

$C_j \in L_i$, we will connect N_{V_i} to N_{C_j} by an undirected edge in $G^{'}$ for all $1 \leq i \leq k$. We can observe that $G^{'}$ is a Bipartite graph. Now it is easy to see that there is a valid k colouring for the given clique iff there is a perfect matching in the Bipartite graph ([56], 7.10.2) $G^{'}$. The problem of finding a perfect matching in a Bipartite graph is well known to be in complexity class P ([22, 8, 35, 37, 20]). Hence the proposition is proved. □

Now we consider the counting problems as follows.

#MULTI CLIQUE-COLOURING:

Input: An undirected graph $G(V,E)$ over n vertices and l disjoint cliques (there is no edge $\{u,v\} \in E$ such that u and v are part of two different cliques) inside the graph, each of size k.

Task: How many valid k colourings are there for the graph ?

Parameter: $(n-kl).log(k)$

It is easy to see that, for the problem of general k colouring of any graph with n vertices, k colouring of that graph can be encoded in $n.log(k)$ bits. So the natural witness length or the parameter of that problem is $n.log(k)$. The given input problem size is clearly polynomially bounded in $n.log(k)$. Hence the problem is trivially in VC_0. Corresponding counting problem is in $\#VC_0$ for the same reason. But for MULTI CLIQUE-COLOURING problem, we prove the following result.

Theorem 31. MULTI CLIQUE-COLOURING *is* VC_2*-hard.*

Proof. We will prove this result by showing a w-reduction from NAE-SAT to MULTI CLIQUE-COLOURING. In chapter 3, we have already proved that NAE-SAT is VC_2-complete (Theorem 12). Hence, this theorem will directly follow from that reduction.

We consider a NAE-SAT instance ϕ over x_1, x_2, ..., x_n. Let us consider c_1, c_2, ..., c_l are the clauses of ϕ. We consider a graph $G(V,E)$ from ϕ as follows. We will introduce following set of vertices.

$V_1 = \{x_i, \bar{x_i} | 1 \leq i \leq n\}$,

$V_2 = \{p_i(l_{i,1}), p_i(l_{i,2}), \ldots, p_i(l_{i,t_i}) | c_i = l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,t_i}, 1 \leq i \leq l\}$,

$V_3 = \{p_i(l_{i,t_i+1}), p_i(l_{i,t_i+2}), \ldots, p_i(l_{i,n}) | c_i = l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,t_i}, t_i < n, 1 \leq i \leq l\}$,

$V_4 = \{v_i | 1 \leq i \leq n-2\}$,

$V_5 = \{n_1, n_2\}$.

$V = \bigcup_{i=1}^{5} V_i$

We can see that total number of vertices are $N = 3n + nl$. Now we will connect these vertices introducing following set of edges.

$E_1 = \{\langle v_i, v_j\rangle | v_i, v_j \in V_4, i \neq j\}$,
$E_2 = \{\langle p_i(l_{i,1}), l_{i,1}\rangle, \langle p_i(l_{i,2}), l_{i,2}\rangle, \ldots, \langle p_i(l_{i,t_i}), l_{i,t_i}\rangle | c_i = l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,t_i}, 1 \leq i \leq l,$
$l_{i,1}, l_{i,2}, \ldots, l_{i,t_i} \in V_1\}$,
$E_3 = \{\langle p_i(l_{i,r}), p_i(l_{i,s})\rangle | 1 \leq r < s \leq n, 1 \leq i \leq l\}$,
$E_4 = \{\langle n_1, v_j\rangle, \langle n_2, v_j\rangle, \langle x_i, v_j\rangle, \langle \bar{x_i}, v_j\rangle, \langle x_i, \bar{x_i}\rangle | v_j \in V_4, 1 \leq j \leq n-2, x_i, \bar{x_i} \in V_1, 1 \leq i \leq$
$n, n_1, n_2 \in V_5\}$,
$E_5 = \{\langle n_1, a\rangle, \langle n_2, a\rangle, \langle n_1, n_2\rangle | \forall a \in V_3, n_1, n_2 \in V_5\}$.
$E = \bigcup_{i=1}^{5} E_i$

Our graph construction is now complete. We are now giving a brief description of what is already written in mathematical notation above. Firstly, without loss of generality we can assume that there is no clause in ϕ such that both x_i and $\bar{x_i}$ are present in the same clause for any $1 \leq i \leq n$. Because, we can remove such literals easily from ϕ in polynomial time. Hence, any clause can contain at most n literals. If we look into the graph construction, for each literal in each clause, we have introduced a vertex in V_2. If any clause contains less than n literals, we have introduced extra vertices for all such clauses to make the vertices corresponding to each clause n. These vertices in V_2 and V_3 are the vertices of l disjoint cliques (there is no edge $\{u, v\} \in E$ such that u and v are part of two different cliques) inside the graph, each of size k.
). It is easy to see that edge set E_3 is constructing l disjoint cliques, one for each each clause, each of size n. There is another clique of size $n-2$ produced by edge set E_1.

Now we claim that ϕ has a satisfying assignment such that each clause of it contains at least one *True* and one *False* literal iff G is n colourable. We prove this claim as follows.

Without loss of generality, we assume that the n colours are $0, 1, \ldots, n-1$. Suppose ϕ has a satisfying assignment such that each clause of it contains at least one *True* and one *False* literal. Now, any such assignment will assign some Boolean value to x_i and $\bar{x_i}$ for all $1 \leq i \leq n$. We will assign colours 0 and 1 to all the vertices in V_1 according to that assigned Boolean values. Colours 2 to $n-1$ will be assigned to the vertices in V_4 such that no two vertices in V_4 get the same colour. Any of the $(n-2)!$ colourings is fine. n_1 and n_2 can take either 0 or 1 as they are connected to all the vertices in V_4. They are also connected to each other. So we will arbitrarily assign 0 to n_1 and 1 to n_2. The opposite assignment is fine as well. Now, as each clause contains at least one *True* and one *False* literal, vertices corresponding to those in V_1 have taken at least one 0 and one 1 colour already. For any clause c_i, $1 \leq i \leq l$, suppose $l_{i,r}$ is *True* and $l_{i,s}$ is *False*, $1 \leq r, s \leq t_i, r \neq s$. So the colour to the vertex corresponding to $l_{i,r}$ (present

in V_1) is 1 and the colour to the vertex corresponding to $l_{i,s}$ (present in V_1) is 0. Now, we will assign colour 0 to $p_i(l_{i,r})$ and 1 to $p_i(l_{i,s})$. Hence, the clique corresponding to clause c_i has two vertices with assigned colours (0 and 1). All its neighbouring vertices are coloured by either 0 or 1. So now we can assign colours to remaining $n-2$ vertices easily. It is now easy to see that we have found a valid colouring when ϕ has a satisfying assignment such that each clause of it contains at least one *True* and one *False* literal.

Now, we are going to prove our claim in opposite direction. Suppose, there is a valid colouring for graph G. So the clique corresponding to vertex set V_4 of size $n-2$ have $n-2$ different colours, one for each vertex. Without loss of generality, we assume that the n colours are $0,1,\ldots,n-1$ and denote the colours assigned to the vertices in V_4 by $2,3,\ldots,n-1$. So clearly, the colours of vertices in V_1 and V_5 are either 0 or 1 as all of them are connected to the vertices in V_4. As a result, we can also see that the vertices in V_3 can not take colour 0 or 1 as they are connected to the vertices in V_5. So, each clique corresponding to each clause has colour 0 and 1 in it for some vertices present only in V_2. Suppose colour 0 is assigned to $p_i(l_{i,r})$ and 1 to $p_i(l_{i,s})$ for some clause c_i, $1 \leq i \leq l$, $1 \leq r,s \leq t_i, r \neq s$. So clearly colour 1 is assigned to the vertex $l_{i,r}$ (present in V_1) and 0 to $l_{i,s}$ (present in V_1). Now, if we take the Boolean values corresponding to the colour assignments for vertices in V_1, we can see that it is a satisfying assignment for ϕ such that each clause of it contains at least one *True* and one *False* literal. Hence our claim is proved.

The parameter for the MULTI CLIQUE-COLOURING problem instance is $(N-ln).log(n)$. Clearly it is polynomially bounded in n (number of input variables in ϕ), the parameter of the NAE-SAT problem instance. Hence, we have found our required w-reduction which proves the theorem. □

We are now going to prove the following result corresponding to the counting version of this problem.

Corollary 6. #MULTI CLIQUE-COLOURING *is* $\#VC_2$*-hard.*

Proof. In Theorem 25, we have already proved that #NAE-SAT is $\#VC_2$-complete. So, to prove this result we are just going show that the reduction presented in the proof of Theorem 31 is weakly w-parsimonious.

We use exactly the same notations here as used in the proof of Theorem 31. Firstly, we want to point out again that the parameter of the problem is $(N-kl).log(k)$ where $k = n$. Because, there is a natural witness of length $(N-kl).log(k)$ bits for this problem.

So now, we are going to find, for any satisfying assignment for ϕ such that each clause of it contains at least one *True* and one *False* literal, how many witnesses are there for the #MULTI CLIQUE-COLOURING problem instance of length $(N-kl).log(k)$ where $k=n$. So we have to find, how many ways we can properly colour vertices in V_1, V_4 and V_5. Firstly, we can see that for any valid (such that each clause of it contains at least one *True* and one *False* literal) satisfying assignment for ϕ, we can choose $n-2$ colours for V_4 from n in $\binom{n}{n-2} = \binom{n}{2}$ ways. Those $n-2$ colours can be given to vertices in V_4 in either of $(n-2)!$ ways. For each such colour assignment, vertices corresponding to positive and negative literals in V_1 can be coloured in $2!$ ways. Finally, for each such colour assignment for V_1 and V_4, vertices in V_5 can be coloured in another $2!$ ways. So, for any valid (such that each clause of it contains at least one *True* and one *False* literal) satisfying assignment for ϕ, total number of witnesses for #MULTI CLIQUE-COLOURING problem instance is $\binom{n}{n-2}(n-2)!2!2! = 2n!$. Hence, total number of witnesses for the #NAE-SAT instance = total number of witnesses for the #MULTI CLIQUE-COLOURING instance divided by $2n!$. Hence, we the reduction is weakly w-parsimonious and #MULTI CLIQUE-COLOURING is $\#VC_2$-hard. □

4.3 Different types of circuit problems with respect to compression

We are now going to define an interesting parametric counting problem #MONOTONE WEIGHTED-CNFSAT and other related problems. Then we are going to prove some completeness and hardness results for all of them.

#MONOTONE WEIGHTED-CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

Similarly we can define:

#ANTIMONOTONE WEIGHTED-CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are negative literals, and another integer k ($\leq n$).

Task:How many satisfying assignments are there for ϕ such that exactly k variables

are assigned to be *True*?

Parameter: $k.log(n)$

#WEIGHTED-HORNSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where each clause contains at most one positive literal and another integer $k \leq n$.

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

It is easy to see that any #ANTIMONOTONE WEIGHTED-CNFSAT instance is a #WEIGHTED-HORNSAT instance, but the converse may not be true.

Similarly, we can define corresponding decision problems. Now we are going to prove the hardness and completeness results corresponding to the above problems. Throughout this discussion, when we say that there is some assignment of *weight k*, we mean that exactly k variables are assigned to be *True*.

Now we are now going to show a strongly parametric w-parsimonious reduction from #WEIGHTED-CNFSAT to #MONOTONE WEIGHTED-CNFSAT. As the counting problem #WEIGHTED-CNFSAT is #$W[2]$-complete from Definition 46 and any #MONOTONE WEIGHTED-CNFSAT instance is also a #WEIGHTED-CNFSAT instance, in this way we can prove that MONOTONE-WEIGHTED-CNFSAT is #$W[2]$-complete. We will first consider the decision version and argue that the reduction works for counting version as well. The result for the decision version is presented in [14] (Theorem 1) which applies the technique used by J. Flum and M. Grohe ([25], Lemma 7.5). We have already used this technique for a different problem in the proof of Proposition 19 (in chapter 3).

Theorem 32. #MONOTONE WEIGHTED-CNFSAT *is* #$W[2]$*-complete.*

Proof. Let ϕ be a Boolean expression in conjunctive normal form consisting of l clauses $C_1, C_2, \ldots, C_l$ with variables $x_1, x_2, \ldots, x_n$. We are going to construct another Boolean formula $\phi^{'}$ from ϕ with $n^{'}$ (polynomially bounded in n) number of variables such that, ϕ is satisfiable with k number of variables assigned to be *True* iff $\phi^{'}$ is also satisfiable with $k^{'}$ (polynomially bounded in k) number of variables assigned to be *True*. In our construction of $\phi^{'}$, it will be conjunction of two different formulae. First formula (ψ) will not be dependent on ϕ, but the second one ($\psi^{'}$) will be. We have described the constructions of these two formulae below.

To construct such a new formula, we introduce two sets of new variables $X_{i,j}$ (for i

$\in [k]$ and $j \in [n]$) and $Y_{i,j,j'}$ (for $i \in [k-1]$ and $1 \leq j < j' \leq n$) with intending meanings as below:

$X_{i,j}$: the i^{th} variable set to be *True* is x_j,

$Y_{i,j,j'}$: the i^{th} variable set to be *True* is x_j and the $(i+1)^{th}$ is $x_{j'}$.

So total number of variables n' are bounded by $(k.n + (k-1)n(n-1)/2)$ which is clearly polynomially bounded in n as $k \leq n$.

We are now going to construct a new Boolean formula as conjunction of k Boolean formulae as follows:

$\psi = \phi_0 \wedge \phi_1 \wedge \ldots \wedge \phi_{k-1} \ldots (i)$

where $\phi_0 = (\bigwedge_{i=1}^{k} \bigvee_{a \in X_i} a) \wedge (\bigwedge_{i=1}^{k-1} \bigvee_{b \in Y_i} b)$, $X_i = \{X_{i,j} | j \in [n]\}$, $Y_i = \{Y_{i,j,j'} | 1 \leq j < j' \leq n\}$

$\phi_i = \bigwedge_{j \in [n]} (\bigvee_{1 \leq j_1 < j_2 \leq n, j_1 \neq j} (X_{i,j} \vee Y_{i,j_1,j_2}) \wedge \bigvee_{1 \leq j_1 < j_2 \leq n, j_2 \neq j} (X_{i+1,j} \vee Y_{i,j_1,j_2}))$, $i = 1, 2, \ldots, k-1$.

We claim that

Lemma 5. *Let l_1, l_2, …, $l_k \in [n]$ be k distinct integers. Any assignment of weight $(2k-1)$ satisfying ψ and setting X_{1,l_1}, X_{2,l_2}, …, X_{k,l_k} to True, must set Y_{1,l_1,l_2}, Y_{2,l_2,l_3}, …, Y_{k-1,l_{k-1},l_k} to True.*

Proof. In ϕ_0, we can see there are exactly $2k-1$ clauses (corresponding to X_i, $1 \leq i \leq k$ and Y_i, $1 \leq i \leq k-1$). All of them are consists of different sets of input variables. So clearly exactly one variable in each X_i and Y_i should be *True* for any assignment of weight $(2k-1)$ satisfying ψ.

Let us now consider that X_{i,l_i} be the variable of X_i set to be *True*. For any such fixed $i \in [k-1]$, let $Y_{i,l,m}$ be the variable of Y_i set to *True*. If $l \neq l_i$, then in ϕ_i the sub-formula $\bigvee_{1 \leq j_1 < j_2 \leq n, j_1 \neq j} (X_{i,j} \vee Y_{i,j_1,j_2})$ will not be satisfied. Similarly if $m \neq l_{i+1}$ the sub-formula $\bigvee_{1 \leq j_1 < j_2 \leq n, j_2 \neq j} (X_{i+1,j} \vee Y_{i,j_1,j_2})$ will not be satisfied. Hence the lemma is proved. □

Now we will replace all the positive literals x_j of the initial formula ϕ by the following disjunction:

$\bigvee_{i \in [k]} X_{i,j}$

and all the negative literals $\bar{x_j}$ by the following formula (which is basically disjunctions of variables):

$\bigvee_{j_1 \in [n], j < j_1} X_{1,j_1} \vee (\bigvee_{i \in [k-1]} \bigvee_{j_1, j_2 \in [n], j_1 < j < j_2} Y_{i,j_1,j_2}) \vee \bigvee_{j_2 \in [n], j > j_2} X_{k,j_2} \ldots (ii)$

to construct the new Boolean formula ψ'.

It is easy to see that x_j is *True* iff exactly one variable in the disjunction $\bigvee_{i\in[k]} X_{i,j}$ is *True*. But if x_j is *False*, i.e., $\bar{x_j}$ is *True*, x_j is either smaller (with respect to the ordering of the variables of ϕ by their indices, $x_1, x_2, \ldots, x_n$) than the first variable set to be *True* or between two consecutive variables set to be *True* or after the last variable set to be *True*. The formula (ii) above captures this.

We present $\phi' = \psi \wedge \psi'$ as our final formula. We can see that in ϕ', all the literals are positive literals. Hence, it is a *Monotone* formula. From the above explanations we can understand that ϕ is satisfiable with k variables assigned to be *True* iff ϕ' is also satisfiable with k' ($k' = 2k-1$) variables assigned to be *True*. The parameter of the initial WEIGHTED-CNFSAT instance is $k.log(n)$ where as the parameter of the final instance is $(2k-1)log(n')$. So clearly it is a w-reduction and it works for counting version as well as the reduction is strongly parsimonious. Hence, MONOTONE WEIGHTED-CNFSAT is $WK[2]$-hard and corresponding counting problem is $\#WK[2]$-hard.

Any MONOTONE WEIGHTED-CNFSAT instance is a WEIGHTED-CNFSAT instance. So clearly MONOTONE WEIGHTED-CNFSAT $\in WK[2]$ (similar to its counting counter-part) and the theorem follows. □

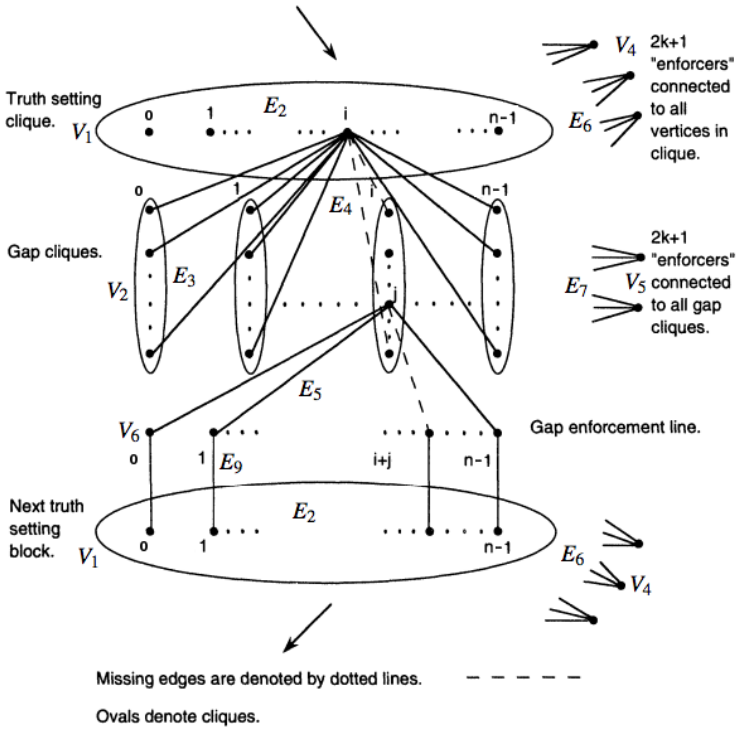
Now, we are going to give an alternate proof of the above theorem. In this proof we will use the technique applied by Downey and Fellows (Theorem 2.1 [42]). Here we will reduced the #WEIGHTED-CNFSAT instance to a #SET COVER and then #HITTING SET instance in the intermediate stage. As a result, it also proves the $\#WK[2]$-completeness of the problems #SET COVER and #HITTING SET. Corresponding results for decision versions are pointed out in [14].

Alternate proof of Theorem 32:

Proof. According to Definition 46, the counting problem #WEIGHTED-CNFSAT is $\#WK[2]$-complete. Hence, we prove the $\#WK[2]$-completeness result of #MONOTONE WEIGHTED-CNFSAT, showing a reduction from a #WEIGHTED-CNFSAT instance to an instance of the problem #MONOTONE WEIGHTED-CNFSAT.

We can observe that, a #MONOTONE WEIGHTED-CNFSAT instance ϕ with n number of input variables is also a #WEIGHTED-CNFSAT instance. Hence it is trivial to prove that #MONOTONE WEIGHTED-CNFSAT $\in \#WK[2]$.

Now we are going to prove the hardness result. There are several steps to prove this result. We are going to show the steps one by one. Let ϕ be a Boolean expression in conjunctive normal form consisting of l clauses $C_1, C_2, \ldots, C_l$ with variables $x_1, x_2,$



$\ldots, x_n$. Downey and Fellows (Theorem 2.1 [42]) have shown the technique to produce a graph $G(V,E)$ from ϕ in polynomial time, such that it has a dominating set of size $2k$ if and only if ϕ is satisfied by a truth assignment of weight k.We are going to use same technique to get a #SET COVER instance from ϕ as described below.

The vertex set V of G is the union of the following sets of vertices:

$V_1 = \{a[r,s] : 0 \leq r \leq k-1, 0 \leq s \leq n-1\},$

$V_2 = \{b[r,s,t] : 0 \leq r \leq k-1, 0 \leq s \leq n-1, 1 \leq t \leq n-k+1\},$

$V_3 = \{c[j] : 1 \leq j \leq l\},$

$V_4 = \{a^{'}[r,u] : 0 \leq r \leq k-1, 1 \leq u \leq 2k+1\},$

$V_5 = \{b^{'}[r,u] : 0 \leq r \leq k-1, 1 \leq u \leq 2k+1\},$

$V_6 = \{d[r,s] : 0 \leq r \leq k-1, 0 \leq s \leq n-1\}.$

The edge set E of G is the union of the following sets of edges.

$E_1 = \{c[j]a[r,s] : X_s \in C_j\},$

$E_2 = \{a[r,s]a[r,s^{'}] : s \neq s^{'}\},$

$E_3 = \{b[r,s,t]b[r,s,t^{'}] : t \neq t^{'}\},$

$E_4 = \{a[r,s]b[r,s^{'},t] : s \neq s^{'}\},$

$E_5 = \{b[r,s,t]d[r,s^{'}] : s^{'} \neq (s+t)(mod\ n)\},$

$E_6 = \{a[r,s]a^{'}[r,u]\},$

$E_7 = \{b[r,s,t]b^{'}[r,u]\},$

$E_8 = \{c[j]b[r,s,t] : \exists i\ \bar{x_i} \in C_j, s < i < (s+t)(mod\ n)\}$,
$E_9 = \{d[r,s]a[r^{'},s] : r^{'} = (r+1)(mod\ n)\}$.

It can be easily seen that $G(V,E)$ has a dominating set of size $2k$ if and only if ϕ is satisfied by a truth assignment of weight k. It is also easy to see that no vertex from set V_3 can be present in dominating set of size $2k$, because of verities in V_4 and V_5.

Now we will construct a SET COVER instance (S,C) from $G(V,E)$. Corresponding to all the vertices in G except those in V_4, we will construct a set containing that vertex with all its neighbouring vertices. Collection of all those sets is C for our SET COVER instance. Set of all the vertices V of G is S for our SET COVER instance. So clearly, SET COVER instance (S,C) has a set cover of size $2k$ if and only if there is a satisfying assignment for ϕ with weight k. This reduction is clearly strongly parsimonious and works for corresponding counting problem as well. Number of sets in C is bounded by $poly(n)$ and hence the parameter for the SET COVER instance is $2k.log(poly(n))$. So clearly the reduction is strongly w-parsimonious.

We can now construct a HITTING SET instance (S_h,C_h) from SET COVER instance (S,C) using the standard well known reduction mentioned in chapter 3 (Proposition 22). This reduction is also strongly w-parsimonious.

So far all the reductions from #WEIGHTED-CNFSAT to #HITTING SET are strongly w-parsimonious. Now we are going to show another strongly w-parsimonious reduction from #HITTING SET instance (S_h,C_h) to a #MONOTONE WEIGHTED-CNFSAT instance.

Now let us try to described the reduction and corresponding function σ and τ. Firstly we describe the function σ. Let us denote the counting problem #HITTING SET as F and #MONOTONE WEIGHTED-CNFSAT as G. σ will take the #HITTING SET instance (S_h,C_h) and corresponding to each element $a_i \in S_h$ it will construct a variable y_i. Corresponding to each set c_j^h in C_h, it will construct horn clause c_j taking $y_t \in c_j$ iff $a_t \in c_j^h$. Now taking conjunction of all these clauses, σ will construct the HornSat instance ψ. It is easy to see that the #HITTING SET instance (S_h,C_h) has a hitting set of size k iff ψ has a satisfying assignment with number of 1 in each assignment equal to k. We will take this ψ as our #MONOTONE WEIGHTED-CNFSAT instance (as all its literals are positive literals) which is asking how many satisfying assignments for ψ are there with weight k. Now we describe the algorithm τ. It will take #HITTING SET instance (S_h,C_h), use σ, G and will return the number of hitting set of size k for the instance (S_h,C_h). Let us denote the instance (S_h,C_h) as x which asks the number of hitting set of size k for the instance (S_h,C_h). Now we can see that $G(\sigma(x))$ is the

number of satisfying assignments of ψ with number of 1 in each assignment equal to k. Clearly it is equal to the number of hitting sets of size k for the #HITTING SET instance (S_h, C_h). τ is clearly running in time polynomial in size of x. We can also see that σ is not increasing the parameter, it is still $k.log(n)$ where n is the number of elements in S_h. Hence, #MONOTONE WEIGHTED-CNFSAT is $\#W[2]$-complete. □

Theorem 33. #ANTIMONOTONE WEIGHTED-CNFSAT *is* $\#VC[2]$*-hard.*

Proof. We will prove this result by showing a w-reduction from SAT to ANTIMONOTONE WEIGHTED-CNFSAT. Then we will argue that the reduction is strongly parsimonious and works for counting version as well. As #SAT is $\#VC[2]$-complete (from Definition 44), this theorem will directly follow from that argument.

To prove this theorem we are going to re-define 2 problems with different parameter as follows.

WEIGHTED-CNFSAT$_n$:

Input: A formula ϕ in conjunctive normal form of size m over n variables, and another integer k ($\leq n$).

Task: Is there any satisfying assignments for ϕ such that exactly k variables are assigned to be *True*?

Parameter: n

MONOTONE WEIGHTED-CNFSAT$_n$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another integer k ($\leq n$).

Task: Is there any satisfying assignments for ϕ such that exactly k variables are assigned to be *True*?

Parameter: n

Previously we considered $k.log(n)$ as the parameter for both the problems. But we know that the number of input variables n is also a natural witness length for these problems. Similarly we can define the counting versions of these problems.

#WEIGHTED-CNFSAT$_n$:

Input: A formula ϕ in conjunctive normal form of size m over n variables, and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: n

#MONOTONE WEIGHTED-CNFSAT$_n$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another integer k ($\leq n$).
Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?
Parameter: n

Now, suppose ϕ is a Boolean formula in *CNF* with n variables. This ϕ is our SAT instance which is asking if there is any satisfying assignment for ϕ. We now convert ϕ to $\phi^{'}$ where $\phi^{'}$ is basically a WEIGHTED-CNFSAT$_n$ instance. So here we ask, if there is any satisfying assignment of weight k for $\phi^{'}$, i.e., exactly k ($\leq n^{'}$, number of variables for $\phi^{'}$) variables are assigned to be *True*. We can use our original reduction to do that as we have used in chapter 3 (Proposition 21). Suppose $x_1, x_2, \ldots, x_n$ are the input variables for ϕ. We will introduce a new set of variables $y_1, y_2, \ldots, y_n$. We will replace each $\bar{x_i}$ in ϕ by y_i, $0 \leq i \leq n$. We rename this new formula as ϕ_y. Now we construct our $\phi^{'}$ as follows:

$\phi^{'} = \phi_y \wedge \bigwedge_{i=1}^{n}((x_i \vee y_i) \wedge (\bar{x_i} \vee \bar{y_i}))$

It is now easy to see that ϕ is satisfiable iff $\phi^{'}$ has a satisfying assignment of weight $k = n$. Number of variables of $\phi^{'}$ is $n^{'} = 2n$. So clearly it is a w-reduction and also works for counting versions.

Now, following similar technique as in the proof of Theorem 32, we reduce this $\phi^{'}$ to a MONOTONE WEIGHTED-CNFSAT$_n$ instance $\phi^{''}$. $\phi^{''}$ is a monotone formula in *CNF* and here we ask if there is any satisfying assignment of weight $k^{''} = 2k^{'} - 1$ for $\phi^{''}$. $k^{''}$ is less than or equal to $n^{''}$, the number of variables of $\phi^{''}$. From the proof of Theorem 32 we can also see that $n^{''} = (k.n^{'} + (k-1)n^{'}(n^{'}-1)/2)$ which is clearly polynomially bounded in $n^{'}$ as $k \leq n^{'}$. Hence, all the reductions we have used so far are strongly parametric w-parsimonious.

Now we will convert this $\phi^{''}$, to our final ANTIMONOTONE WEIGHTED-CNFSAT instance ψ as follows. We will just invert all the literals in $\phi^{''}$ to their negations and construct a new formula ψ. Now for this new reduced instance, we ask, if there is any satisfying assignment for ψ such that $(n^{''} - k^{''})$ variables are assigned to be *True*. We take $(n^{''} - k^{''}).log(n^{''})$ as the parameter for this ANTIMONOTONE WEIGHTED-CNFSAT instance ψ. We can see that $(n^{''} - k^{''}).log(n^{''})$ is polynomially bounded in $n^{''}$. It is now easy to see that if any assignment, say $b_1 b_2 \ldots b_n$ ($b_i \in \{0,1\}$, $1 \leq i \leq n$) satisfies $\phi^{''}$, $\bar{b_1}\bar{b_2}\ldots\bar{b_n}$ will satisfy ψ. $\bar{b_1}\bar{b_2}\ldots\bar{b_n}$ contains exactly $(n^{''} - k^{''})$ 1s in the binary string of length $n^{''}$ iff $b_1 b_2 \ldots b_n$ contains $k^{''}$ 1. So clearly, we have found a w-reduction from SAT instance ϕ to ANTIMONOTONE WEIGHTED-CNFSAT instance

ψ which is strongly parsimonious and works for both decision and counting versions. Hence, #ANTIMONOTONE WEIGHTED-CNFSAT is #$VC[2]$-hard. □

As any ANTIMONOTONE WEIGHTED-CNFSAT instance is also a WEIGHTED-CNFSAT instance, it is easy to see that #ANTIMONOTONE WEIGHTED-CNFSAT $\in$ $WK[2]$. As any #ANTIMONOTONE WEIGHTED-CNFSAT problem instance is also a #WEIGHTED-HORNSAT instance (where each clause contains at most one positive literals), similar proof as Theorem 33 works for #WEIGHTED-HORNSAT as well. Similarly, this problem is present in $WK[2]$ as well.

We can also consider slightly different version of the previous two problems as follows:

#ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are negative literals and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that number of variables assigned to be *True* $\leq k$?

Parameter: $k.log(n)$

It is easy to see that decision versions of the above problem is easy to solve as we can set all the variables *False*. So any such problems are trivially solvable and hence in VC_0. But for counting versions, both the problems $\in$ #$WK[2]$ and are #VC_2-complete.

Theorem 34. #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ *is* #VC_2*-hard and present in* #$WK[2]$.

Proof. We will prove this result by showing a parametric w-T-reduction from the counting problem #ANTIMONOTONE WEIGHTED-CNFSAT to the problem #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ and vice versa. We denote the counting problem #ANTIMONOTONE WEIGHTED-CNFSAT as F, and the counting problem #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ as G. Suppose ϕ is a F instance and the counting problem asks how many satisfying assignments are there for ϕ of weight k. We denote this counting problem as $(\phi, k.log(n))$. To reduce this problem to a #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ instance, our counting reduction algorithm σ will trivially output the same instance ϕ. But the output counting problem is asking how many satisfying assignments of ϕ are there of weight at most (instead of exactly) k.

We will now construct τ (the algorithm with an oracle for G) to get the total number of satisfying assignments of the F instance from G instance. τ will consider two different F instances $(\phi, k.log(n))$ and $(\phi, (k-1).log(n))$ give $(G(\sigma(\phi, k.log(n))) -$

$G(\sigma(\phi,(k-1).log(n))))$ as output. $G(\sigma(\phi,(k-1).log(n))$ gives total number of satisfying assignments of ϕ of weight at most $k-1$ and $G(\sigma(\phi,k.log(n)))$ gives the same of weight at most k. So clearly if we subtract the first from the second, we will get total number of satisfying assignments of ϕ of weight exactly k. Hence the algorithm τ is correct and the counting problem #ANTIMONOTONE WEIGHTED-CNFSAT is parametric w-T-reducible to #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$.

To show the reduction in the opposite direction, suppose ψ is a #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ instance and the counting problem asks how many satisfying assignments are there for ψ of weight at most k. In this reduction, our σ will be same as before, i.e., it will trivially output the same instance ψ. But the output counting problem is asking how many satisfying assignments of ψ are there of weight exactly (instead of at most) k.

Now we are going to construct τ (the algorithm with an oracle for F) to get the total number of satisfying assignments of the G instance from F instance. τ will consider $k+1$ different G instances $(\phi,0.log(n))$ (it is asking for satisfying assignments of weight 0), $(\phi,1.log(n))$, ..., $(\phi,k.log(n))$ and give $\sum_{t=0}^{k}(F(\sigma(\phi,t.log(n)))$ as output. $F(\sigma(\phi,t.log(n))$ gives total number of satisfying assignments of ϕ of weight exactly t. So clearly $\sum_{t=0}^{k}(F(\sigma(\phi,t.log(n)))$ gives total number of satisfying assignments of ϕ of weight at most k i.e., $G(\phi,k.log(n))$. Hence, #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ is w-T-reducible to the counting problem #ANTIMONOTONE WEIGHTED-CNFSAT and the proof of the theorem completes. □

Now we can similarly define following problem and prove the results related to them as mentioned.

#MONOTONE WEIGHTED-CNFSAT$_{\leq}$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that number of variables assigned to be *True* $\leq k$?

Parameter: $k.log(n)$

Corollary 7. #MONOTONE WEIGHTED-CNFSAT$_{\leq}$ *is* #$W[2]$*-complete.*

Above corollary can be proved similarly by showing parametric w-T-reductions to and from the problems #MONOTONE WEIGHTED-CNFSAT.

We can consider the following problem as well:

#WEIGHTED-HORNSAT$_{\leq}$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where each clause contains at most one positive literal and another integer $k \leq n$.

Task: How many satisfying assignments are there for ϕ of weight $\leq k$?

Parameter: $k.log(n)$

As any #ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$ problem instance is also a #WEIGHTED-HORNSAT$_{\leq}$ instance, #WEIGHTED-HORNSAT$_{\leq}$ is $\#VC_2$-hard (Theorem 34) as well. It is not surprising that, corresponding decision version, WEIGHTED-HORNSAT$_{\leq}$ is in P, same as ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$. But the proof is not so trivial as ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$. We have briefly mentioned this proof below.

Proposition 30. WEIGHTED-HORNSAT$_{\leq}$ *is in* VC_0.

Proof. We are going to describe a polynomial time algorithm to prove this. Suppose ϕ is a Horn Boolean formula presented as WEIGHTED-HORNSAT$_{\leq}$ instance. So each clause of ϕ contains at most one positive literal. We have to check if there is any satisfying assignment of ϕ of weight at most k. Firstly we will check if there is any clause containing single positive literal in the formula. If so, we will pick any such clause and assign corresponding variable to *True* and remove that clause from ϕ. If the same variable is present in any other clause as positive literal we will remove that clause from ϕ as well. If the same variable is present in any clause as negative literal, we will remove only that literal from that clause. If that negative literal is the only literal in the clause, we will stop and declare that ϕ is not satisfiable. Otherwise we will check if the new formula still contains any clause containing any single positive literal. If so we will continue like above until the formula doesn't contain any single positive literal clause. When all the single positive literal clauses are removed (new single positive literal clauses, not present in ϕ, can arise in intermediate steps, we have to remove all of them), we will just count how many variables of ϕ are assigned to *True* so far. If that count is greater than k, we will declare that there is no satisfying assignment of ϕ of weight at most k. Otherwise, we will assign all the remaining variables *False* and can see that we have already found a satisfying assignment of ϕ of weight at most k. The runtime of the algorithm is polynomially bounded in number of variables of ϕ as at each step we will assign exactly one variable to *True* until all the single literal clauses are removed. Hence, WEIGHTED-HORNSAT$_{\leq}$ is in VC_0. □

Now we are going to consider the un-weighted version of these problems and see

where they stand in $\#VC$ hierarchy. But before that, we are going to prove a very simple result. Let us consider the following problems.

#DEPTH$_i$OR CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where the top level Boolean operation is *OR*.

Task: How many satisfying assignments are there for C ?

Parameter: n

Similarly we can define:

#DEPTH$_i$AND CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where the top level Boolean operation is *AND*.

Task: How many satisfying assignments are there for C ?

Parameter: n

Similarly we can define the counting versions of these two problems. They are already defined in [13]. It is clear that both of these problems are present in $\#VC_i$. Analogous result in decision version is also true. It is also proved in [13] that, for $i \geq 3$, DEPTH$_i$AND CIRCUITSAT is VC_i-complete but DEPTH$_i$OR CIRCUITSAT $\in$ VC_{i-1} (Claim 2.16, [13]). But for counting versions, that proof does not work. So, we prove the following result here.

Theorem 35. *For $i \geq 3$, both* #DEPTH$_i$AND CIRCUITSAT *and* #DEPTH$_i$OR CIRCUITSAT *are $\#VC_i$-complete.*

Proof. To prove this result we have to consider both the problems one by one. So we consider #DEPTH$_i$AND CIRCUITSAT ($i \geq 3$) first. For that we have to show w-T-reductions to and from #DEPTH$_i$CIRCUITSAT (Definition 44). We know that any #DEPTH$_i$AND CIRCUITSAT instance C is after all a circuit of depth at most i. So it is trivial that #DEPTH$_i$AND CIRCUITSAT $\in \#VC_i$ ($i \geq 3$).

To prove the hardness result, we show a reduction from #DEPTH$_i$CIRCUITSAT to #DEPTH$_i$AND CIRCUITSAT. Suppose one circuit C of depth at most i ($i \geq 3$) is given to us with n input variables. If the top level Boolean operation in C is already *AND*, we do not need to do anything else. Because, in that case it is already a #DEPTH$_i$AND CIRCUITSAT instance. But if the top level Boolean operation is *OR*, we just take $C^{'} = \bar{C}$ as our final #DEPTH$_i$AND CIRCUITSAT instance. It is easy to see that the top level operation of $C^{'}$ is *AND* (applying De Morgan's law) when the top level operation of C is *OR*.

We can also see that total number of satisfying assignments of $C = 2^n$ - total number of satisfying assignments of $C^{'}$. Because, any assignment that will make $C^{'}$ unsatisfiable, will satisfy C. Hence, clearly we have found a weakly w-parsimonious reduction from #DEPTH$_i$CIRCUITSAT to #DEPTH$_i$AND CIRCUITSAT and conclude that #DEPTH$_i$AND CIRCUITSAT is VC_i-complete.

The proof for VC_i-completeness of #DEPTH$_i$OR CIRCUITSAT is very much the same. The containment result is similarly trivial to see. Hardness proof is also similar, except, in this case if the top level operation is not *OR* then only we will take the complement. Hence, For $i \geq 3$, both #DEPTH$_i$AND CIRCUITSAT and #DEPTH$_i$OR CIRCUITSAT are #VC_i-complete. □

Now we define the following problems.

#DEPTH$_i$ MONOTONE CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where all the literals at the bottom level are positive literals.

Task: How many satisfying assignments are there for C ?

Parameter: n

Similarly we can define:

#DEPTH$_i$ ANTIMONOTONE CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where all the literals at the bottom level are negative literals.

Task: How many satisfying assignments are there for C ?

Parameter: n

It is easy to see that decision versions of both the above problems are easy to solve. We can set all the variables to Boolean *True* (or *False*) to make any DEPTH$_i$ MONOTONE CIRCUITSAT (or DEPTH$_i$ ANTIMONOTONE CIRCUITSAT) instance satisfiable. So finding one witness is easy. Now we are going to prove some interesting results corresponding to their counting versions.

Theorem 36. #DEPTH$_i$ MONOTONE CIRCUITSAT *is #VC_i-complete for any* $i \geq 3$.

Proof. We show a parametric w-T-reduction from #DEPTH$_i$AND CIRCUITSAT to #DEPTH$_i$ MONOTONE CIRCUITSAT to prove this theorem. We use F to denote the counting problem #DEPTH$_i$AND CIRCUITSAT and G to denote #DEPTH$_i$ MONOTONE CIRCUITSAT. Let us consider that C is a Boolean circuit of depth at most i ($\geq$ 3) over n variables. This C is our #DEPTH$_i$AND CIRCUITSAT instance. The top level gate is *AND* gate. If we consider the bottom level literals of C, it can contain both

positive and negative literals. Suppose the variables of C are $x_1, x_2, \ldots, x_n$. We now introduce another set of n variables, $y_1, y_2, \ldots, y_n$. Now we will replace any literal $\bar{x_i}$ present in C by y_i. If for any $j \in \{1, 2, \ldots, n\}$, $\bar{x_j}$ is not present at all in C, we can ignore corresponding y_j variable. Let us consider this new circuit is $C^{'}$ and also consider that S ($\subseteq \{1, 2, \ldots, n\}$) is the set such for any $i \in S$, $\bar{x_i}$ is present in C. After that, we will construct $C^{''}$ from $C^{'}$ as follows:

$C^{''} = C^{'} \wedge \bigwedge_{i \in [n], i \in S}(x_i \vee y_i)$

In our *w-T*-reduction, σ will construct this new circuit $C^{''}$ from C in polynomial time. It is clear that $C^{''}$ is a #DEPTH$_i$ MONOTONE CIRCUITSAT instance as all the literals are positive literals here. Now we have to construct τ which can use (C, n), σ and G and should give total number of witnesses of the counting problem #DEPTH$_i$AND CIRCUITSAT instance (C, n) as output.

τ (the algorithm with an oracle for G) will use σ to construct $C^{''}$ from C, and then construct another circuit $C^{'''}$ from it as follows:

$C^{'''} = C^{''} \wedge \bigvee_{i \in [n], i \in S}(x_i \wedge y_i)$

We assume $n^{'''}$ to be the number of variables of $C^{'''}$. We can see that $C^{'''}$ is still a #DEPTH$_i$ MONOTONE CIRCUITSAT instance for any $i \geq 3$. After constructing this new circuit, τ will give following as output:

$G(\sigma(C, n))$ - $G(C^{'''}, n^{'''})$

To prove that the algorithm τ is correct, we have to prove that $F(C, n) = G(\sigma(C, n))$ - $G(C^{'''}, n^{'''})$. It is easy to see that, total number of satisfying assignments of C is equal to the total number of satisfying assignments of $C^{'} \wedge \bigwedge_{i \in [n], i \in S}(x_i \vee y_i) \wedge \bigwedge_{i \in [n], i \in S}(\bar{x_i} \vee \bar{y_i})$. But this circuit is not Monotone though $C^{''}$ is. But clearly, total number of satisfying assignments of $C^{''}$ is $\geq$ that of C. So we have to subtract the extra assignments to get our result. That is why $C^{'''}$ is constructed as $C^{''} \wedge \overline{\bigwedge_{i \in [n], i \in S}(\bar{x_i} \vee \bar{y_i})}$ which is same as $C^{''} \wedge \bigvee_{i \in [n], i \in S}(x_i \wedge y_i)$ as mentioned above. $C^{'''}$ is a #DEPTH$_i$ MONOTONE CIRCUITSAT instance for any $i \geq 3$. Hence, the algorithm τ is correct.

Total number of variables of all the circuits are polynomially bounded by n. So clearly it is a *w-T*-reduction. Any #DEPTH$_i$ MONOTONE CIRCUITSAT instance is also a #DEPTH$_i$CIRCUITSAT instance. Hence #DEPTH$_i$ MONOTONE CIRCUITSAT $\in \#VC_i$ and the theorem follows. □

Theorem 37. #DEPTH$_i$ ANTIMONOTONE CIRCUITSAT *is* $\#VC_i$*-complete for any* $i \geq 3$.

Proof. We will prove this result by showing a strongly *w*-parsimonious reduction from

#DEPTH$_i$ MONOTONE CIRCUITSAT to #DEPTH$_i$ ANTIMONOTONE CIRCUITSAT and vice versa. This reduction actually works for $i = 2$ as well. Let us consider that C is a Boolean circuit of depth at most i over n variables. This C is our #DEPTH$_i$ MONOTONE CIRCUITSAT instance. The top level gate is *AND* gate. If we consider the bottom level literals of C, it contains only positive literals. Suppose the variables of C are $x_1, x_2, \ldots, x_n$. We now introduce another set of n variables, $y_1, y_2, \ldots, y_n$. Now we will replace any literal x_i present in C by $\bar{y_i}$ and call this new circuit as $C^{'}$. If $X_{||n||}$ denotes binary string of length n it is easy to see that for any assignment $X_{||n||}$ that satisfies C, $\overline{X_{||n||}}$ (this is bit by bit complementation of the binary string $X_{||n||}$) satisfies $C^{'}$. So clearly total number of satisfying assignments of C is same as total number of satisfying assignments of $C^{'}$. Total number of variables are also same for both the circuits. Hence this reduction is strongly parsimonious.

The reduction in the opposite direction is actually very similar. We can similarly replace any negative literal $\bar{z_i}$ of some #DEPTH$_i$ ANTIMONOTONE CIRCUITSAT instance D by y_i and construct a new circuit $D^{'}$. $D^{'}$ is a #DEPTH$_i$ MONOTONE CIRCUITSAT instance which has same number of satisfying assignments as D. Hence, #DEPTH$_i$ MONOTONE CIRCUITSAT and #DEPTH$_i$ ANTIMONOTONE CIRCUITSAT are equivalent to each other with respect to strongly parametric w-parsimonious reduction and #DEPTH$_i$ ANTIMONOTONE CIRCUITSAT is #VC_i-complete for any $i \geq 3$. □

Next, consider another interesting problem.

#OR-SAT:

Input: m Boolean formulae ϕ_1 , ... , ϕ_m, each with n input variables and of size at most $poly(n)$.

Task: How many assignments to the input variables are there such that for each assignment, there exists $i \in [m]$ such that ϕ_i is satisfiable ?

Parameter: $(n + log(m))$

In the above problem, all the Boolean formulae ϕ_1 , ... , ϕ_m, are consists of different sets of input variables. That is why the parameter is taken as $(n + log(m))$ as in this case, any witness will encode the satisfiable smaller formula in $log(m)$ bits and then represent the truth assignments of its input variables in n bits. Even if they are sharing some common variables, we can still encode the natural witness in $(n + log(m))$ bits. So it is a legitimate choice of parameters for this problem. Harnik and Naor [13] have already proved that, decision version of this problem is VC_{or}-complete and Fortnow

and Santhanam [32] have proved that this problem is unlikely to be in VC_0 unless polynomial hierarchy collapses. But if we consider the counting version, we can see following result.

Theorem 38. #OR-SAT *is in* $\#VC_0$.

Proof. We will prove this result by showing a Turing compression algorithm for the #OR-SAT counting problem. Let us consider ψ consists of a disjunction of m Boolean formulae $\phi_1, \ldots, \phi_m$, each with n input variables and of size at most $poly(n)$. This ψ is our #OR-SAT instance. Let us denote the counting problem #OR-SAT as F and the #SAT counting problem (parameter n, the number of input variables) as G. F and G take instance of corresponding counting problem as input and give total number of witnesses of that instance as output. To compress any #OR-SAT instance, we will find a pair of polynomial time computable algorithms (σ,τ). σ will take any #OR-SAT instance as input and give a Boolean formula in CNF as output. But the output formula will be polynomially bounded in $(n+log(m))$, the parameter of the #OR-SAT problem. On the other hand, τ can use ψ, G and σ and gives total number of witnesses of ψ as output. We also have to make sure that within τ, whenever we are using G to find the total number of witnesses of any #SAT instance, that instance size should be polynomially bounded in the parameter of ψ.

We now describe our σ and τ (the algorithm with an oracle for G). If ψ contains exactly one small Boolean formula (of size $poly(n)$) ϕ_1 with n input variables, it is already compressed. So in that case σ will output only ϕ_1. Otherwise, σ takes ψ as input and gives any trivial Boolean formula of size $poly(n+log(m))$ as output. On the other hand, τ will take ψ and split it into individual smaller CNF Boolean formulae $\phi_1, \ldots, \phi_m$. Then it will give $\sum_{i=1}^{m} G(\sigma((\phi_i,n))$ as output. It is easy to see that $\sum_{i=1}^{m} G(\sigma((\phi_i,n))$ is exactly the total number of witnesses of ψ. We can also see that, within τ, whenever we are using G to find the total number of witnesses of any #SAT instance ϕ_i, that instance size is polynomially bounded in n. Hence our σ and τ are legitimate compression algorithm pair and #OR-SAT is in $\#VC_0$. □

Although the above proof is quite simple, the implication is quite interesting. Here we are trying to develop a theory of compressibility analogous to the theory of solvability. In theory of solvability, we know that counting problems are always difficult to solve compared to its decision version. If we can find the solution of any counting problem, we can check if it is greater than 0 or not. If yes, we know that corresponding

decision version is a *yes* instance, other wise *no* instance. But in theory of compressibility, the same thing is not true. Theorem 38 is giving one such counter example where the decision version is difficult to compress but the counting version is easy to compress.

Now we define a few more interesting problems.

SELECTED DOMINATING SET:

Input: A graph $G(V,E)$ with n vertices, and a subset $N \subseteq V$.

Task: Is there a subset $N^{'} \subseteq N$, such that it is a dominating set of graph G and every vertex in N is dominated by exactly one vertex in $N^{'}$?

Parameter: $|N|$

Above, we can see that the problem is defining a special kind of dominating set with respect to some other vertex set N. We refer them as *selected dominating set*.

SELECTED SET COVER:

Input: A set S with $|S| = n$, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ and a subset $N \subseteq S$.

Task: Is there a subset $C^{'} \subseteq C$, such that it is a Set Cover of the set S and every element in N is covered by exactly one set in $C^{'}$?

Parameter: m

SELECTED HITTING SET:

Input: A set S with $|S| = n$, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ and a subset $N \subseteq C$.

Task: Is there a subset $S^{'} \subseteq$ of S, such that it is a Hitting Set of C and every set in N is hit by exactly one element in $S^{'}$?

Parameter: n

Similarly we can define the term *selected set cover* and *selected hitting set*. The problems defined above are actually combination of general SET COVER (or DOMINATING SET/HITTING SET) and corresponding exact versions of them. We have already seen in previous chapter that EXACT COVER and EXACT HITTING SET are VC_1-hard. With different parameter they are actually VC_E-complete. On the other hand SET COVER, DOMINATING SET and HITTING SET are VC_2-complete and present in VC_3. Hence, it is interesting to define the selected versions of these problems and check how they behave with respect to compression.

We are now going to prove the following theorem.

Theorem 39. SELECTED HITTING SET *is* VC_2*-complete.*

Proof. We already know that SAT is VC_2-complete [13]. So we will prove this result by showing *w*-reductions from SAT to SELECTED HITTING SET and vice versa.

Hardness Result:

To prove the hardness result, we have to show a *w*-reduction from SAT to SELECTED HITTING SET. We will show this reduction step by step. Firstly we will show a *w*-reduction from SAT to SELECTED DOMINATING SET. This part of the proof is inspired by the technique used by Downey and Fellows (Theorem 2.1 [42]). We have already used that in the alternate proof of Theorem 32. Here, we are going to generalize that technique.

Let ϕ be a Boolean expression in conjunctive normal form consisting of l clauses $C_1, C_2, \ldots, C_l$ with variables $x_1, x_2, \ldots, x_n$. Now we are going to construct a graph G from ϕ such that ϕ is satisfiable iff G has an *Selected dominating set*. The graph construction is going to be similar to what we have seen in the alternate proof of Theorem 32. In that proof, for a given k, we constructed a gadget. Now, for for all $k = 0,1, \ldots, n$, we are going to construct separate gadgets similar to that. For $k = 2,\ldots, n$, gadget construction will be exactly similar. So let us describe that first. For $k = 2,\ldots, n$,we will construct sub-graph G_k with vertices V_k and edges E_k, as follows.

The vertex set V_k of G_k is the union of the following sets of vertices:

$V_1^k = \{a[r,s] : 0 \leq r \leq k-1, 0 \leq s \leq n-1\}$,

$V_2^k = \{b[r,s,t] : 0 \leq r \leq k-1, 0 \leq s \leq n-1, 1 \leq t \leq n-k+1\}$,

$V_3^k = \{a^{'}[r,u] : 0 \leq r \leq k-1, 1 \leq u \leq 2k+1\}$,

$V_4^k = \{b^{'}[r,u] : 0 \leq r \leq k-1, 1 \leq u \leq 2k+1\}$,

$V_5^k = \{d[r,s] : 0 \leq r \leq k-1, 0 \leq s \leq n-1\}$.

The edge set E_k of G is the union of the following sets of edges.

$E_1^k = \{a[r,s]a[r,s^{'}] : s \neq s^{'}\}$,

$E_2^k = \{b[r,s,t]b[r,s,t^{'}] : t \neq t^{'}\}$,

$E_3^k = \{a[r,s]b[r,s^{'},t] : s \neq s^{'}\}$,

$E_4^k = \{b[r,s,t]d[r,s^{'}] : s^{'} \neq (s+t)(mod\ n)\}$,

$E_5^k = \{a[r,s]a^{'}[r,u]\}$,

$E_6^k = \{b[r,s,t]b^{'}[r,u]\}$,

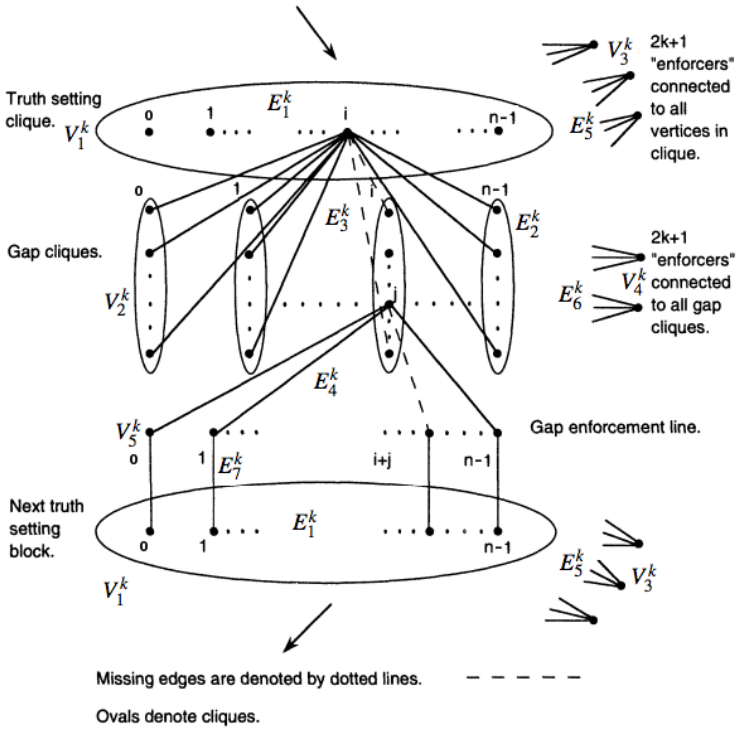
$E_7^k = \{d[r,s]a[r^{'},s] : r^{'} = (r+1)(mod\ n)\}$.

For any such fixed k and fixed r, the graph is shown in the picture.

Besides, we will add some extra vertices and edges as follows.

$V^{'} = \{c[j] : 1 \leq j \leq l\}$

$E_1^{'} = \{c[j]a[r,s] : X_s \in C_j$ for all $a[r,s]$ in $V_1^k, 2 \leq k \leq n\}$,



$E_2^{'} = \{c[j]b[r,s,t] : \exists i\ \bar{x_i} \in C_j, s < i < (s+t)(mod\ n)$, for all $b[r,s,t]$ in $V_2^k, 2 \leq k \leq n\}$.

For $k = 0$, V_0 is just one vertex. This vertex will be connected to $c[j] : 1 \leq j \leq l$, if $\exists i\ \bar{x_i} \in C_j$.

For $k = 1$,

$V_1^1 = \{a[s] : 0 \leq s \leq n-1\}$,

$V_2^1 = \{b[s] : 0 \leq s \leq n-1\}$,

$V_3^1 = \{a^{'}[u] : 1 \leq u \leq 2n+1\}$,

$V_4^1 = \{b^{'}[u] : 1 \leq u \leq 2n+1\}$,

$E_1^1 = \{a[s]a[s^{'}] : s \neq s^{'}\}$,

$E_3^1 = \{a[s]b[s^{'}] : s \neq s^{'}\}$,

$E_5^1 = \{a[s]a^{'}[u]\}$,

$E_6^1 = \{b[s]b^{'}[u]\}$.

Now to connect these vertices with $V^{'}$, similarly we will connect $c[j]a[s] : X_s \in C_j$ for all $a[s] \in V_1^1$. We also connect $c[j]b[s]$ if $\exists i\ \bar{x_i} \in C_j$ and $i \neq s$ for all $b[s] \in V_2^1$.

There will be another set of vertices and edges as follows (this part is working as weight controller):

$U_1 = \{q[s] : 0 \leq s \leq n\}$,

$U_2 = \{q^{'}[t] : 1 \leq t \leq 2n+1\}$,

$E_1^q = \{q[s]q[s^{'}] : s \neq s^{'}\}$,

$E_2^q = \{q[s]q^{'}[t]\}$,
$E_3^q = \{q[s]a : \forall a \in V_k, 0 \leq k \leq n, k \neq s\}$.

So, as we can see above, we have connected $q[s]$ to all the V_k vertices except when $k = s$. Besides, we have used another set of $2n+1$ vertices (U_2) which are connected to all the $q[s] : 0 \leq s \leq n$. This part is ensuring that exactly one vertex from U_1 is always selected.

Now our graph construction is complete. Combining all the smaller graphs mentioned above, we will get our desired graph G where N is all the vertices except $V^{'}$. So clearly number of vertices in N is polynomially bounded in n. Now, it is not that difficult to see that ϕ is satisfiable iff G has a *Selected dominating set* where parameter is $|N|$. Let us explain why this claim correct. Suppose ϕ is satisfiable. So there is some satisfying assignment of weight k, $0 \leq k \leq n$. We will select $u[k]$ from U_1 as one vertex in our final dominating set. So, all the vertices except those in V_k and $V^{'}$ are covered. Now, we can not select any other vertex from anywhere except those in vertex set V_k. After that, depending upon the assignment, correct vertices will be selected from V_k as shown by Downey and Fellows (Theorem 2.1 [42]). Clearly, all the vertices except those in $V^{'}$, will be dominated by exactly one vertex. The proof in the opposite direction is also similar. Suppose there is a desired *Selected dominating set* for G. So clearly exactly one vertex, say $u[k]$, from U_1 is selected. Now remaining selection of vertices will map one satisfying assignment of weight k for ϕ.

Hence, we have proved that SELECTED DOMINATING SET is VC_2-hard. Now we are going to show another w-reduction from SELECTED DOMINATING SET to SELECTED SET COVER.

We will construct a SELECTED SET COVER instance (S,C) from $G(V,E)$ as follows. Corresponding to all the vertices in N, we will construct a set containing that vertex with all its neighbouring vertices (i.e., all the vertices connected directly with that vertex). Collection of all those sets is C for our SELECTED SET COVER instance. Set of all the vertices of G is the set of elements S and set of vertices N is same $N \subseteq S$ for our SELECTED SET COVER instance. So clearly, SELECTED SET COVER instance (S,C) has a set cover such that the elements corresponding to N is covered exactly once if and only if G has a *Selected dominating set*. The parameter of both the problems are clearly polynomially bounded. Thus, we have proved that SELECTED SET COVER is VC_2-complete.

Now, we are going to reduce this SELECTED SET COVER instance (S,C) to our final SELECTED HITTING SET instance (S_h,C_h) to prove the hardness result. The

reduction is quite simple. Corresponding to each set $s_i \in C$, we will construct one element for S_h. Corresponding to each element $a_i \in S$, we will construct a set c_i for C_h. c_i will contain all those elements of S_h corresponding to all the sets in C where a_i is present. We also construct $N_h = \{c_i | a_i \in N\}$. So clearly $N_h \subseteq C_h$. It is now easy to see that (S,C) has a *selected set cover* with respected to N iff (S_h,C_h) has a *selected hitting set* with respect to N_h. Clearly the parameter is also polynomially bounded and we have proved that SELECTED HITTING SET is VC_2-hard.

Membership Result:

To prove the membership, we are going show a w-reduction from SELECTED HITTING SET to SAT. Let us consider a SELECTED HITTING SET instance (S_h,C_h). We are now going to construct a Boolean formula $\phi^{'}$ which will be in CNF. We will just construct one clause corresponding to each set in C_h such that $a \in s_t$ for some $s_t \in C_h$, $a \in c_t$ where c_t is a clause (disjunction of literals) for $\phi^{'}$. All the elements in S_h will be the variable of $\phi^{'}$. We can now notice that (S_h,C_h) has a *selected hitting set* with respect to N_h iff $\phi^{'}$ has a satisfying assignment such that each of the clauses corresponding to the sets in N_h has exactly one literal assigned to be *True*. Suppose $\phi^{'}$ is of following format:

$\phi^{'} = \bigwedge_{i=0}^{l-1} C_i \wedge \bigwedge_{i=0}^{l^{'}-1} D_i^{'}$

So $\phi^{'}$ has $l + l^{'}$ clauses (C_i, $0 \leq i \leq l-1$ and $D_i^{'}$, $0 \leq i \leq l^{'}-1$) and (S_h,C_h) has a *selected hitting set* with respect to N_h iff $\phi^{'}$ has a satisfying assignment such that exactly one literal in every C_i, $0 \leq i \leq l-1$, is assigned to be *True*. Number of variables of $\phi^{'}$ is same as the parameter of the SELECTED HITTING SET instance. But this $\phi^{'}$ is not our final SAT instance. To obtain our final SAT instance, we consider that:

$C_i = \bigvee_{j=0}^{i_t-1} l_{i,j}$, $0 \leq i \leq l-1$.

It means, for $0 \leq i \leq l-1$, any clause C_i is disjunctions of i_t literals.

Now we construct $C_i^{'}$ corresponding to each C_i, $0 \leq i \leq l-1$ as follows.

$C_i^{'} = C_i \wedge \bigwedge_{0 \leq p < q \leq (i_t-1)} (\bar{l_{i,p}} \vee \bar{l_{i,q}})$.

It is easy to see that $C_i^{'}$ is satisfiable iff exactly one literal in C_i is assigned to be *True*.

Now we construct the new formula $\phi^{'}$ as follows.

$\phi^{''} = \bigwedge_{i=0}^{l-1} C_i^{'} \wedge \bigwedge_{i=0}^{l^{'}-1} D_i^{'}$

It is easy to see that $\phi^{''}$ is satisfiable iff $\phi^{'}$ is satisfiable such that exactly one literal in each C_i, $0 \leq i \leq l-1$ is assigned to be *True*. The number of variables (parameter) in both $\phi^{'}$ and $\phi^{''}$ are same. So clearly we have found a w-reduction from SELECTED HITTING SET to SAT. Hence, SELECTED HITTING SET is VC_2-complete. □

From the above proof we can see that we have already proved the following corollary.

Corollary 8. SELECTED DOMINATING SET *and* SELECTED SET COVER *are* VC_2*-complete.*

The counting versions of these problems are defined below.

#SELECTED DOMINATING SET:

Input: A graph $G(V,E)$ with n vertices, and a subset $N \subseteq V$.

Task: How many subsets of N are there, such that each of them is a dominating set of graph G and every vertex in N is dominated by exactly one vertex ?

Parameter: $|N|$

#SELECTED SET COVER:

Input: A set S with $|S| = n$, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ and a subset $N \subseteq S$.

Task: How many subsets of C are there, such that each of them is a Set Cover of the set S and every element in N is covered by exactly one set ?

Parameter: m

#SELECTED HITTING SET:

Input: A set S with $|S| = n$, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ and a subset $N \subseteq C$.

Task: How many subsets of S are there, such that each of them is a Hitting Set of C and every set in N is hit by exactly one element ?

Parameter: n

As all the reductions in the proof of Theorem 39 are strongly parsimonious and work for counting versions as well, we can now prove the following corollary.

Corollary 9. #SELECTED HITTING SET, #SELECTED DOMINATING SET *and* #SELECTED SET COVER *are* #VC_2*-complete.*

4.4 Counting complexity class #VC_E and related problems

We have already introduced the new class VC_E in chapter 3 (section 3.3). We are now going to consider the counting counterpart of that problem.

#EXACT CNF-SAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly one literal in each clause of ϕ is assigned to be *True* ?

Parameter: n

We are now defining the counting complexity class $\#VC_E$ as follows:

Definition 53. *$\#VC_E$ is the class of counting problems which are w-T-reducible to* #EXACT CNF-SAT.

We have already proved that $VC_E \subseteq VC_1$ in chapter 3 (Theorem 16). It is easy to see that corresponding reductions are eventually strongly w-parsimonious and hence works for counting problems as well. As a result, we can conclude that,

Theorem 40. $\#VC_E \subseteq \#VC_1$

We are now going to consider some counting problems, decision versions of which are already considered in chapter 3 (section 3.3).

#EXACT HITTING SET$_n$:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$.

Task: How many exact hitting sets (a subset $S \subseteq V$ such that $|S \cap e| = 1$ for all $e \in E$) are there for (V, E) ?

Parameter: n

#EXACT COVER$_n$:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set).

Task: How many exact covers (a subset $S \subseteq E$ of pairwise disjoint sets with $\bigcup S = V$) are there for (V, E) ?

Parameter: m

#EXACT TEST SET:

Input: A set S with $|S| = n$, another set $T \subseteq Pair(S)$ where $Pair(S)$ denotes all the distinct pairs of elements from S, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ ($\wp()$ denotes the power set).

Task: How many subsets $C' \subseteq C$ are there, such that for every distinct pair u and v from T, there exists exactly one set c in C' such that either of u and v is present in c, but not both ?

Parameter: m

In chapter 3, we have already proved that corresponding decision versions are VC_E-complete (Theorem 17 and 18). It is easy to see that corresponding reductions are strongly w-parsimonious and hence we can conclude that,

Theorem 41. #EXACT HITTING SET$_n$, #EXACT COVER$_n$ *and* #EXACT TEST SET *are #VC_E-complete.*

In chapter 3, we have mentioned about a direct w-reduction from EXACT CNF-SAT to SAT (Proposition 17). It is easy to see that same reduction works for counting problems as well as corresponding reduction is strongly w-parsimonious.

Let us now define some counting problems that we wish to consider in this section. We have already seen slightly different version of these problems before (Theorem 30). There we actually considered the weighted version of these problems. Now we are considering more general versions.

#KNAPSACK$_n$:

Input: T is a set of n items as follows, $T = \{b_1, b_2, \ldots, b_n\}$. $f : b_i \to \mathbb{N}$ is a function mapping each item to some natural number. Another integer A is given.

Task: How many binary strings x of length n ($x = x_1x_2\ldots x_n$, $x_i \in \{0,1\}$) are there such that $\Sigma_{i=1}^{n} f(b_i)x_i \leq A$?

Parameter: n

#SUBSET SUM$_n$:

Input: T is a set of n items as follows, $T = \{b_1, b_2, \ldots, b_n\}$. $f : b_i \to \mathbb{N}$ is a function mapping each item to some natural number. Another integer A is given.

Task: How many binary strings x of length n ($x = x_1x_2\ldots x_n$, $x_i \in \{0,1\}$) are there such that $\Sigma_{i=1}^{n} f(b_i)x_i = A$?

Parameter: n

It is easy to see that the decision version of #KNAPSACK$_n$ problem is easy to solve in polynomial time. We can just test the smallest value (some $f(b_j)$) in the given set whether it is smaller than or equal to A or not. If yes, we will set corresponding bit x_j to 1 and make all the remaining bits to 0. So we have just found one witness and we are done. If no, there is no such witness for this problem. Hence it is solvable in $O(n)$ and hence it is in VC_0. But for counting version, we can find the following result.

Theorem 42. #KNAPSACK$_n$ *is #VC_E-hard.*

Proof. We will prove this result step by step. We will first show that SUBSET SUM$_n$ is VC_E-hard. Then we will argue that corresponding reduction is strongly w-parsimonious

and hence it works for counting version as well, i.e., #SUBSET SUM$_n$ is #VC_E-hard. After that we will show a w-T-reduction from #SUBSET SUM$_n$ to #KNAPSACK$_n$.

To show that SUBSET SUM$_n$ is VC_E-hard, we will show a reduction from EXACT CNF-SAT to SUBSET SUM$_n$. This reduction technique is inspired by the standard polynomial reduction from 3-SAT to SUBSET SUM ([39, 15]). Although, that original reduction is not a w-reduction. Suppose an EXACT CNF-SAT instance ϕ is given to us with n variables (x_1, x_2, …, x_n) and l clauses. To construct a SUBSET SUM$_n$ instance from ϕ, we construct $2n$ items, 2 for each variable x_i , one corresponding to x_i itself (denoted as b_i) and another one corresponding to $\bar{x_i}$ (denoted as $\bar{b_i}$). The function f will map each item to a natural number, represented as $(n+l)$ bits long string as follows. For each b_i and $\bar{b_i}$, i^{th} bit is set to 1, i = 1, 2, …, n. If x_i is present in j^{th} clause, $(n+j)^{th}$ bit corresponding to b_i is set to 1. Similarly, if $\bar{x_i}$ is present in t^{th} clause, $(n+t)^{th}$ bit corresponding to $\bar{b_i}$ is set to 1. Rest of the bits are set to 0. This technique is represented in following table.

Table 4.1: Knapsack construction from ϕ

Item	x_1	x_2	…	x_n	c_1	c_2	…	c_l
b_1	1	0	…	0	0	1	…	0
$\bar{b_1}$	1	0	…	0	1	0	…	1
b_2	0	1	…	0	0	0	…	1
$\bar{b_2}$	0	1	…	0	0	1	…	0
…	…	…	…	…	…	…	…	…
b_n	0	0	…	1	0	0	…	0
$\bar{b_n}$	0	0	…	1	0	0	…	1
A	1	1	…	1	1	1	…	1

In this table, we have assumed that the clause c_2 of formula ϕ contains x_1, clause c_1 and c_l contain $\bar{x_1}$ and so on. Let us take any value $b > 2n$. We consider each of those binary strings mentioned above, are actually b-ary strings. So they are actually represented in base b and hence the value corresponding to any string $s_1 s_2 \ldots s_{n+l}$ ($s_i \in \{0,1\}$, i = 1, 2, …, $n+l$) is the natural number $\sum_{i=0}^{n+l-1} s_{i+1} b^i$. We choose A for our SUBSET SUM$_n$ instance, to be the string of length $n+l$ containing all 1. This is also represented in base b.

Now suppose ϕ has any satisfying assignment such that exactly one literal in every clause is assigned to *True*, and corresponding assignment is represented as $y_1 y_2 \ldots y_n$

($y_i \in \{0,1\}$, $i = 1, 2, \ldots, n$). Then it is clear that $\Sigma_{i=1}^{n} f(b_i)y_i + \Sigma_{i=1}^{n} f(\bar{b_i})\bar{y_i} = A$ and thus we can find our $2n$ bits binary string for the SUBSET SUM$_n$ problem. Similarly, if there exists some binary string $z_1\bar{z_1}z_2\bar{z_2}\ldots z_n\bar{z_n}$ ($z_i \in \{0,1\}$, $i = 1, 2, \ldots, n$) such that $\Sigma_{i=1}^{n} f(b_i)z_i + \Sigma_{i=1}^{n} f(\bar{b_i})\bar{z_i} = A$, we can see that $z_1z_2\ldots z_n$ is a satisfying assignment for ϕ such that exactly one literal in every clause is assigned to *True*. We can see that to obtain a sum A, we can not choose both b_i and $\bar{b_i}$ for any $i = 1, 2, \ldots, n$ as the i^{th} bit of both $f(b_i)$ and $f(\bar{b_i})$ is 1. The base b is chosen large enough such that there is no overflow in the summation. That is why $z_1\bar{z_1}z_2\bar{z_2}\ldots z_n\bar{z_n}$ is written above instead of more general $z_1z_2\ldots z_{2n}$. The parameter of the initial EXACT CNF-SAT instance is n and SUBSET SUM$_n$ is $2n$. So clearly it is a w-reduction. We can also see that this reduction is strongly w-parsimonious and hence works for counting version as well. Hence, #SUBSET SUM$_n$ is VC_E-hard.

Now to show a w-T-reduction from #SUBSET SUM$_n$ to the counting problem #KNAPSACK$_n$ we can use similar technique as used in the proof of Theorem 30. In that proof we have shown a w-T-reduction from #SUBSET SUM to the counting problem #KNAPSACK. But exactly same technique works for the un-weighted version as well. Hence #KNAPSACK$_n$ is #VC_E-hard. □

We now consider another interesting problem MAX CUT$_n$. We have already considered a different version of it in chapter 3 (section 3.2.2).

MAX CUT$_n$:

Input: A weighted graph G with m edges and n vertices, an integer A.

Task: Decide whether there exists a cut in G of size A.

Parameter: n

In the above definition the parameter is n as here any witness can be represented by a binary string of length n, 1 or 0 corresponding to each vertex. 1 represents that corresponding vertex is in set 1 and 0 in other set. In this way the vertex set of the graph can be split into two sets and we can consider the cross edges to check if the *cut* is of size A or not. We can also define the counting version corresponding to this problem and denote as #MAX CUT$_n$.

Theorem 43. MAX CUT$_n$ *is* VC_E*-hard.*

Proof. To show that MAX CUT$_n$ is VC_E-hard, we will show a reduction from EXACT CNF-SAT to MAX CUT$_n$. Suppose an EXACT CNF-SAT instance ϕ is given to us with n variables ($x_1, x_2, \ldots, x_n$) and l clauses. We assume that the size of ϕ is m and

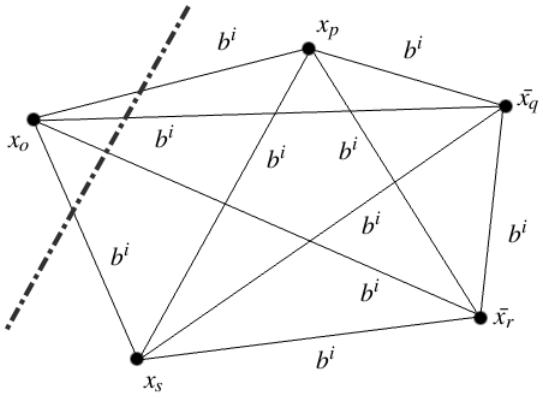


Figure 4.1: Clause-clique corresponding to clause $C_i = x_o \vee x_p \vee \bar{x_q} \vee \bar{x_r} \vee x_s$

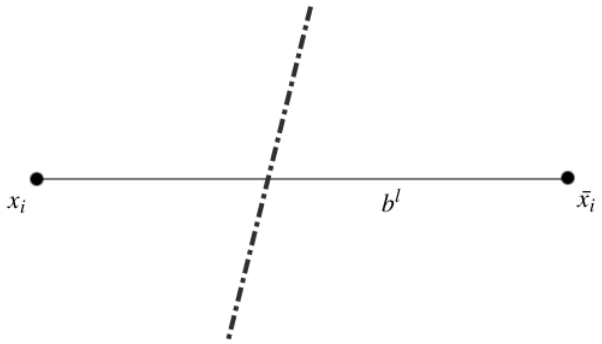


Figure 4.2: Corresponding to pair x_i and $\bar{x_i}$

also assume that its clauses are $C_0, C_1, \ldots, C_{l-1}$. To construct a MAX CUT$_n$ instance from ϕ, we construct a graph G as follows.

G has $2n$ vertices, corresponding to each possible literal: $x_1, x_2, \ldots, x_n, \bar{x_1}, \bar{x_2}, \ldots, \bar{x_n}$. For each clause with t distinct literals, we just make a clique of size t in G connecting the corresponding literal vertices. For any clause C_i $(0 \leq i \leq l-1)$, we set the weight for each edge in corresponding clique to b^i, where b an integer sufficiently large (any integer just greater than m, e.g. $(m+1)$ will be fine). If any edge is present in more than one clause-clique, clearly the weights will be assigned more than once for that edge. For any such edge, where the weights are assigned for than once, we will just add all the weights corresponding to that edge and provide the summation as the final weight.

We also connect all the pairs of vertices corresponding to x_i and $\bar{x_i}$ $(1 \leq i \leq n)$ by

an edge of weight b^l. As x_i and $\bar{x_i}$ $(1 \leq i \leq n)$ can not be present in a single clause of ϕ (if they are present, we can easily remove them doing some pre-processing in $poly(m)$ time), these edges will not be present in any clause-clique. We do not need to bother about clause with single literal as we can remove such clauses by pre-processing the formula in $poly(m)$ time, forcing them to be *True* and propagate corresponding changes through out the formula ϕ. Suppose $t_0, t_1, \ldots, t_{l-1}$ are the number of literals in clauses $C_0, C_1, \ldots, C_{l-1}$ of ϕ respectively (after any pre-processing when required). Now we set the cut size of our MAX CUT$_n$ instance A as $\sum_{i=0}^{l-1}(t_i - 1).b^i + n.b^l$. As the weights of every edge and A are at most exponential of $poly(m)$ where the exponent is also at most $poly(m)$, we can easily represent them in $poly(m)$ bits (taking logarithm).

Now we are going to show why the above construction is correct. Suppose ϕ has a satisfying assignment such that exactly one literal in each clause is *True*. Any such assignment will make each vertex either 0 or 1, corresponding to their logical value. We claim that if all such 0-valued vertices are put into one set, and all the 1-valued vertices into another set, the cut size will be exactly A. It is easy to see that in any clause C_i $(0 \leq i \leq l-1)$, when exactly one literal is true, it will cut exactly $t_i - 1$ edges, each of weight b^i (at least b^i, because one edge may be present in more than one clause-clique). So that will contribute to $(t_i - 1).b^i$ towards A, for all i, 0 to $(l-1)$. As for any pair x_i and $\bar{x_i}$ $(1 \leq i \leq n)$, exactly one of them is *True*, all those vertices will contribute exactly $n.m^l$ towards A. If we add them up, we can see that the cut size becomes $A = \sum_{i=0}^{l-1}(t_i - 1).b^i + n.b^l$. Hence, if ϕ has a satisfying assignment such that exactly one literal in each clause is *True*, G has a *cut* of size exactly A.

To prove it for opposite direction, we first like to point out that A can be written as a b-ary string of length $(l+1)$ as follows, $(t_0 - 1)(t_1 - 1)\ldots(t_{l-1} - 1)n$. Here each bit position corresponds to each clause except the last one, which corresponds to set of pairs. Now if G has a cut of size exactly A, it must cut every clause-clique cutting exactly $t_i - 1$ edges, $0 \leq i \leq l-1$ and cut every pair x_i and $\bar{x_i}$ $(1 \leq i \leq n)$. As b is sufficiently large (b is always greater than total number of literals in any clause and total number of pairs), there is no possibility of overflow when we will add the weights to get A. Hence, if G has a *cut* of size exactly A, ϕ has a satisfying assignment such that exactly one literal in each clause is *True*.

Here the parameter of the MAX CUT$_n$ instance is the number of vertices in G, which is $2n$. It is clearly polynomially bounded by the parameter of the EXACT CNF-SAT instance n. Thus, MAX CUT$_n$ is VC_E-hard. □

Corollary 10. #MAX CUT$_n$ *is* #VC_E*-hard.*

In the proof of Theorem 43, it is easy to see that the reduction is strongly *w*-parsimonious and hence works for counting version. Thus the above corollary is correct. In this context I would like to mention that if we consider the same MAX CUT$_n$ problem for un-weighted graph G, corresponding problem is in VC_0. Analogous result for the counting version is also true. When the graph is un-weighted, we can anyway encode whole graph and the cut size in $poly(n)$, as number of edges are always polynomial in n. So clearly both decision and counting versions are compressible.

We are going to define another interesting problem now.

EXACT TOUR$_n$:

Input: A weighted graph G with m edges and n vertices, an integer A.

Task: Decide whether there exists a tour in G of total cost A.

Parameter: n

Similar problem is already defined in [43]. Adapting their work and modifying it according to our requirement, we are now going to prove the following theorem.

Theorem 44. EXACT TOUR$_n$ *is* VC_E*-hard.*

Proof. In Theorem 42, we have already proved that SUBSET SUM$_n$ is VC_E-hard. Here, we are going to show a reduction from SUBSET SUM$_n$ to EXACT TOUR$_n$ to prove this theorem.

Suppose T, a set of n items ($T = \{b_1, b_2, \ldots, b_n\}$) given to us. $f : b_i \to \mathbb{N}$ is a function mapping each item to some natural number. Another integer A is also given. This SUBSET SUM$_n$ instance asks, is there any binary string x of length n ($x = x_1x_2 \ldots x_n$, $x_i \in \{0,1\}$), such that $\Sigma_{i=1}^{n} f(b_i)x_i = A$? We construct a graph G from T as follows.

For each b_i we have two vertices p_i and q_i. Join p_i and q_i with an edge of weight $f(b_i)$. For i not equal to j, we join p_i and q_j by two edges (that means, by inserting another intermediate vertex). We put the weight $+1$ for one of those edges and -1 for another one. Now we ask if G has a tour of total cost A? It is now easy to see that T has a subset sum A iff G has a tour of total cost A Hence, EXACT TOUR$_n$ is VC_E-hard. □

Here, we would like to mention that the above reduction is not parsimonious. So it does not work for counting version. For any specific witness for SUBSET SUM$_n$ instance, there can be $k!$ witness for the EXACT TOUR$_n$ problem where k is the number of items contributing to the subset sum in that specific witness. But this k is not fixed and dependent on specific witness. That is why the above proof can not say if #EXACT TOUR$_n$ is #VC_E-hard or not.

4.5 Exact Satisfiability Problems

We are now consider some more interesting circuit satisfiability problems. In this case we are basically generalising the notion of EXACT CNF-SAT problem. Firstly, let us consider the following problem.

EXACT DNF-SAT:

Input: A formula ϕ in disjunctive normal form of size m over n variables.

Task: Is there any satisfying assignment for ϕ such that exactly one clause in ϕ is satisfiable ?

Parameter: n

#EXACT DNF-SAT:

Input: A formula ϕ in disjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly one clause in ϕ is satisfiable ?

Parameter: n

We first consider the decision version. After that we will see what we can say about its counting counterpart.

Theorem 45. EXACT DNF-SAT *is* VC_2*-complete.*

Proof. We know that SAT is VC_2-complete [13]. To prove this theorem, we are going to show a w-reduction, from SAT to EXACT DNF-SAT and vice versa.

Hardness Result:

Suppose a SAT instance ϕ is given to us with n variables (x_1, x_2, ..., x_n) and l clauses. We assume that its clauses are C_0, C_1, ..., C_{l-1}. We will construct an EXACT DNF-SAT instance from ϕ in two steps.

We first define another problem DNF-UNSAT and show a w-reduction from SAT to this problem.

DNF-UNSAT:

Input: A Boolean formula ψ in disjunctive normal form of size m over n variables.

Task: Is there any un-satisfying assignment for ϕ ?

Parameter: n

Reduction from SAT to DNF-UNSAT is very simple. We just take $\psi = \bar{\phi}$ as our reduced instance. It is easy to see that ψ is a Boolean formula in disjunctive normal form. Besides, any assignment that will satisfy ϕ, will make ψ unsatisfiable. Similarly, any assignment that will make ψ unsatisfiable, will satisfy ϕ. Number of variables, i.e., the parameter of both the problems are same, n. So clearly it is a w-reduction.

Now we are going to reduce this DNF-UNSAT instance ψ to an EXACT DNF-SAT instance ψ'. We denote the clauses of ψ (here the clauses are conjunction of literals) as $C'_0, C'_1, \ldots, C'_{l-1}$ and construct ψ' as follows.

$\psi' = \bigvee_{i=0}^{l-1}(C'_i \wedge y) \vee y$, where y is an extra variable for ψ' other than $x_1, x_2, \ldots, x_n$.

Hence, ψ' has an extra input variable y and an extra clause containing just y. We denote this extra clause as C'_l. Now it is easy to see that if ψ has an unsatisfying assignment, ψ' has an assignment such that only the last clause C'_l is satisfiable. Because, we can just keep the assignments for $x_1, x_2, \ldots, x_n$ in ψ' same as ψ and set $y = 1$.

To prove it in the opposite direction, suppose that ψ' has exactly one clause satisfiable. We claim that, in that case only C'_l can be satisfiable, and other clauses will be unsatisfiable. It can be proved by contradiction. Suppose that some clause C'_j $(0 \leq j \leq l-1)$ is satisfiable and all other clauses, including C'_l, are unsatisfiable. As C'_j is satisfiable, all its literals are assigned to *True*. We know that y is also a literal in C'_j. So y must be assigned to 1. But it will make C'_l satisfiable as well, making more than one clause in ψ' satisfiable. It is a contradiction. Hence, ψ' has exactly one clause satisfiable iff C'_l is the only clause which is satisfiable. As all other clauses are unsatisfiable and $y = 1$, at least one of the original literals in C'_j $(0 \leq j \leq l-1)$ from C_j must be assigned to *False* or logical 0. So clearly, any such assignment (that will make exactly one of the clauses in ψ' satisfiable) for the input variables in ψ' will assign Boolean values to $x_1, x_2, \ldots, x_n$ in such a way that ψ is unsatisfiable.

In the above discussion, it is also easy to see that the parameter for the new instance ψ' is increased only by one. So clearly it is a w-reduction and SAT is w-reducible to EXACT DNF-SAT. Hence, EXACT DNF-SAT is VC_2-hard.

Membership Result:

To prove the membership result, we have to reduce an EXACT DNF-SAT instance to a SAT instance. Suppose a EXACT DNF-SAT instance ϕ (in disjunctive normal form) is given to us with n variables $(x_1, x_2, \ldots, x_n)$ and l clauses. We assume that its clauses (which are disjunctions of literals) are $C_0, C_1, \ldots, C_{l-1}$. We first construct ϕ' from ϕ as follows.

$\phi' = \phi \wedge \bigwedge_{0 \leq i < j \leq l-1}(\bar{C}_i \vee \bar{C}_j)$ ... Equation (i)

When ϕ has an *exact* satisfying assignment, clearly all the disjunctions, $(\bar{C}_i \vee \bar{C}_j)$ $(0 \leq i < j \leq l-1)$ will be satisfiable. Hence, ϕ' will be satisfiable. Similarly, when ϕ' is satisfiable, if at least two clauses, say C_i and C_j are satisfiable, $(\bar{C}_i \vee \bar{C}_j)$ will be unsatisfiable. So exactly one of $C_0, C_1, \ldots, C_{l-1}$ should be satisfiable. Hence, ϕ has a satisfying assignment such that exactly one of its clauses is satisfiable iff ϕ' has a

satisfying assignment.

But $\phi^{'}$ is not a SAT instance. Because $\phi^{'}$ is a depth 3 Boolean formula. So we have to reduce the depth by one to keep it in conjunctive normal form. In $\phi^{'}$, we can see that ϕ is in *DNF* and if we can convert it into *CNF*, whole $\phi^{'}$ will be in *CNF*. We are going to do that as follows.

We are going to introduce $t = log(l)$ new variables for that. We have already used similar strategy in the proof for Lemma 2 in chapter 2 (similar tricks are used in [13], Claim 2.16, as well). If l is not a power of 2, t will be fraction. But we will always take the ceiling of that. Basically, we can always make l power of 2 by introducing some extra (less than double) empty clauses and then we can follow the same technique below. Suppose these t variables are y_0, y_1, ..., y_{t-1}. It is easy to see that, we can we encode any value from 0 to $l-1$ in t bit long binary string. Suppose $b_0^j b_1^j \ldots b_{t-1}^j$ is such binary encoding for j ($0 \leq j \leq l-1$). We also assume that $l_{0,j}$, $l_{1,j}$, ..., $l_{t_j,j}$ are the literals of clause C_j ($0 \leq j \leq l-1$). Now we modify each literal $l_{r,j}$ in clause C_j ($0 \leq r \leq t_j$, $0 \leq j \leq l-1$) as follows:

$l^{'}_{r,j} = l_{r,j} \bigvee_{i=0}^{t-1}(y_i \oplus b_i^j)$, $0 \leq j \leq l-1$, where $y_i \oplus b_i^j = y_i$ when $b_i^j = 0$ and $y_i \oplus b_i^j = \bar{y_i}$ when $b_i^j = 1$.

We also write,

$C^{'}_j = \bigwedge_{r=0}^{t_j} l^{'}_{r,j}$, $0 \leq j \leq l-1$

We construct $\psi = \bigwedge_{i=0}^{l-1} C^{'}_j$. It is easy to see that ψ is in *CNF*. If ϕ is satisfiable, at least one of the clauses C_0, C_1, ..., C_{l-1} will be satisfiable. Then, if the binary encoding of the index of any such satisfiable clause is assigned to the y-variables of ψ with the same assignments for the x-variables, ψ will be satisfiable. Similarly, when ψ is satisfiable, y variables are assigned to the index of one of the clauses C_0, C_1, ..., C_{l-1}. For that particular clause (say j^{th} clause), $\bigvee_{i=0}^{t-1}(y_i \oplus b_i^j)$ will be *False*, but for any other clause $C_{j^{'}}$, $j^{'} \neq j$, $\bigvee_{i=0}^{t-1}(y_i \oplus b_i^{j^{'}})$ will be *True*. But $l^{'}_{r,j}$ must be *True*, for $0 \leq r \leq t_j$. It implies, $l_{r,j}$ will be *True*, for $0 \leq r \leq t_j$. So now the x variables will make both $C^{'}_j$ and C_j satisfiable making ϕ satisfiable. We have now proved, ϕ is satisfiable iff ψ is satisfiable. Number of variables in ψ is also polynomially bounded, as t is polynomially bounded in n. So it is a w-reduction.

Now replacing ϕ by ψ we modify our $\phi^{'}$ as follows (from equation (i)):

$\phi^{''} = \psi \wedge \bigwedge_{0 \leq i < j \leq l-1}(\bar{C_i} \vee \bar{C_j})$

Clearly $\phi^{'}$ is satisfiable iff $\phi^{''}$ is satisfiable. But now $\phi^{''}$ is in *CNF* and also have number of variables polynomially bounded in n. That is why, $\phi^{''}$ is our final SAT instance. Hence, ϕ has a satisfying assignment such that exactly one of its clauses

is satisfiable iff ϕ'' has a satisfying assignment. It proves that EXACT DNF-SAT $\in$ VC_2. Combining both the parts, we can finally conclude that EXACT DNF-SAT is VC_2-complete. □

If we compare, EXACT CNF-SAT is VC_E-complete hence within VC_1, but EXACT DNF-SAT is VC_2-complete. Now if we consider the counting version, we obtain the following corollary.

Corollary 11. #EXACT DNF-SAT *is* #VC_2*-hard and within* #VC_3.

Proof of the above corollary directly follows from Theorem 45. We can see that the reduction for the hardness proof for Theorem 45 is actually strongly parsimonious. Hence #EXACT DNF-SAT is #VC_2-hard. In the containment proof, we have firstly constructed a DEPTH$_3$CIRCUITSAT instance from the EXACT DNF-SAT instance. This reduction is strongly parsimonious which proves that #EXACT DNF-SAT $\in$ #VC_3. But the remaining part of the reduction is not parsimonious, and hence we can not say that #EXACT DNF-SAT $\in$ #VC_2.

4.6 Appendix: Definitions of the parametric counting problems

List of parametric counting problems we have considered here:

#Φ-WSAT:

Input: A formula ϕ over n variables of size m where $\phi \in \Phi$.

Parameter: $k.log(n)$

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True* ?

#ANTIMONOTONE WEIGHTED-CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are negative literals, and another integer k ($\leq n$).

Task:How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

#ANTIMONOTONE WEIGHTED-CNFSAT$_{\leq}$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are negative literals and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that number of variables assigned to be *True* $\leq k$?

Parameter: $k.log(n)$

#CLIQUE:

Input: A graph G with n vertices, and an integer k.

Task: How many cliques of size k (a pairwise adjacent subset of k vertices) are there in G ?

Parameter: $k.log(n)$

#DEPTH$_i$AND CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where the top level Boolean operation is *AND*.

Task: How many satisfying assignments are there for C ?

Parameter: n

#DEPTH$_i$ ANTIMONOTONE CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where all the literals at the bottom level are negative literals.

Task: How many satisfying assignments are there for C ?

Parameter: n

#DEPTH$_i$CIRCUITSAT:

Input: A circuit C of size m and depth at most i over n variables, $i \geq 2$.

Task: How many satisfying assignments are there for C ?

Parameter: n

#DEPTH$_i$ MONOTONE CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where all the literals at the bottom level are positive literals.

Task: How many satisfying assignments are there for C ?

Parameter: n

#DEPTH$_i$OR CIRCUITSAT:

Input: A Boolean circuit C of depth at most i over n variables where the top level Boolean operation is *OR*.

Task: How many satisfying assignments are there for C ?

Parameter: n

#DNF:

Input: A formula ϕ in disjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ ?

Parameter: n

#EXACT CNF-SAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly one literal in each clause of ϕ is assigned to be *True* ?

Parameter: n

#EXACT COVER$_n$:

Input: A set V with $|V| = n$, a family of sets E with $|E| = m$ where $E \subseteq \wp(V)$ ($\wp()$ denotes the power set).

Task: How many exact covers (a subset $S \subseteq E$ of pairwise disjoint sets with $\bigcup S = V$) are there for (V, E) ?

Parameter: m

#EXACT DNF-SAT:

Input: A formula ϕ in disjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly one clause in ϕ is satisfiable ?

Parameter: n

#EXACT HITTING SET$_n$:

Input: A hypergraph (V, E) with $|V| = n$ and $|E| = m$.

Task: How many exact hitting sets (a subset $S \subseteq V$ such that $|S \cap e| = 1$ for all $e \in E$) are there for (V,E) ?

Parameter: n

#EXACT TEST SET:

Input: A set S with $|S| = n$, another set $T \subseteq Pair(S)$ where $Pair(S)$ denotes all the distinct pairs of elements from S, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ ($\wp()$ denotes the power set).

Task: How many subsets $C^{'} \subseteq C$ are there, such that for every distinct pair u and v from T, there exists exactly one set c in $C^{'}$ such that either of u and v is present in c, but not both ?

Parameter: m

#K-UNIQUE COMPLETE-CYCLESET:

Input: A directed graph $G(V,E)$ with n vertices and m edges and an integer $K \leq m$.

Parameter: $K.log(m)$

Task: How many Unique complete-cycleSet are there in the graph containing at most K unique-cycles ?

#KNAPSACK:

Input: An integer k, a set T of m non-negative integers, and another integer A.

Task: How many sub-sets of k integers of T are there, elements of which sum up to less than or equal to A ?

Parameter: $k.log(m)$

#KNAPSACK$_n$:

Input: T is a set of n items as follows, $T = \{b_1, b_2, \ldots, b_n\}$. $f : b_i \rightarrow \mathbb{N}$ is a function mapping each item to some natural number. Another integer A is given.

Task: How many binary strings x of length n ($x = x_1x_2 \ldots x_n$, $x_i \in \{0,1\}$) are there such that $\Sigma_{i=1}^{n} f(b_i)x_i \leq A$?

Parameter: n

#LOCALCIRCUITSAT:

Input: A string x of length m and a circuit C over ($n + n.log\ m$) variables and of size $O(\ n + n.log\ m\ )$.

Task: How many lists I of n locations in x is there such that $C(\langle x(I), I\rangle) = 1$?

Parameter: ($n + n.log\ m$)

#MONOTONE WEIGHTED-CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

#MONOTONE WEIGHTED-CNFSAT$_{\leq}$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that number of variables assigned to be *True* $\leq k$?

Parameter: $k.log(n)$

#MONOTONE WEIGHTED-CNFSAT$_n$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where all the literals in the formula are positive literals, and another integer k ($\leq n$).

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: n

#MULTI CLIQUE-COLOURING:

Input: An undirected graph $G(V,E)$ over n vertices and l disjoint cliques (there is no edge $\{u,v\} \in E$ such that u and v are part of two different cliques) inside the graph, each of size k.

Task: How many valid k colourings are there for the graph ?

Parameter: $(n-kl).log(k)$

#MULTICOLOURED CLIQUE:

Input: A graph $G = (V, E)$ with $|V| = n$, an integer k and a colouring function $c : V \rightarrow [k]$.

Task: How many multicoloured cliques of size k (a clique containing exactly one vertex of each colour) are there in G ?

Parameter: $k.log(n)$

#NAE-SAT:

Input: A formula ϕ in *CNF* with n variables.

Task: How many satisfying assignments are there for ϕ such that each clause of ϕ contains at least one *True* and one *False* literal ?

Parameter: n

#SAT:

Input: A formula ϕ in CNF with n variables.

Task: How many satisfying assignments are there for ϕ ?

Parameter: n

#SELECTED DOMINATING SET:

Input: A graph $G(V,E)$ with n vertices, and a subset $N \subseteq V$.

Task: How many subsets of N are there, such that each of them is a dominating set of graph G and every vertex in N is dominated by exactly one vertex ?

Parameter: $|N|$

#SELECTED HITTING SET:

Input: A set S with $|S| = n$, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ and a subset $N \subseteq C$.

Task: How many subsets of S are there, such that each of them is a Hitting Set of C and every set in N is hit by exactly one element ?

Parameter: n

#SELECTED SET COVER:

Input: A set S with $|S| = n$, a family of sets C with $|C| = m$ where $C \subseteq \wp(S)$ and a subset $N \subseteq S$.

Task: How many subsets of C are there, such that each of them is a Set Cover of the set S and every element in N is covered by exactly one set ?

Parameter: m

#SET SPLITTING:

Input: A set S of n elements and a collection $\mathcal{C}$ ($|\mathcal{C}| = m$) of subsets of S.

Task: How many ways the set S can be partitioned into 2 subsets s_1 and s_2 ($s_1 \bigcup s_2 = S$ and $s_1 \bigcap s_2 = \phi$), such that none of the sets in $\mathcal{C}$ is contained in either of s_1 and s_2 ?

Parameter: n

#SUBSET SUM:

Input: An integer k, a set S of m non-negative integers, and another integer B.

Task: How many sub-sets of k integers of S are there, elements of which sum up to exactly B ?

Parameter: $k.log(m)$

#SUBSET SUM$_n$:

Input: T is a set of n items as follows, $T = \{b_1, b_2, \ldots, b_n\}$. $f : b_i \to \mathbb{N}$ is a function mapping each item to some natural number. Another integer A is given.

Task: How many binary strings x of length n ($x = x_1x_2\ldots x_n$, $x_i \in \{0,1\}$) are there such that $\Sigma_{i=1}^{n} f(b_i)x_i = A$?

Parameter: n

#VERTEX COVER:

Input: A graph $G(V,E)$ with n vertices, and an integer k.

Task: How many vertex covers of size k are there in G?

Parameter: $k.log(n)$

#WEIGHTED-CNFSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

#WEIGHTED-DNFSAT:

Input: A formula ϕ in disjunctive normal form of size m over n variables.

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

#WEIGHTED-HORNSAT:

Input: A formula ϕ in conjunctive normal form of size m over n variables where each clause contains at most one positive literal and another integer $k \leq n$.

Task: How many satisfying assignments are there for ϕ such that exactly k variables are assigned to be *True*?

Parameter: $k.log(n)$

#WEIGHTED-HORNSAT$_{\leq}$:

Input: A formula ϕ in conjunctive normal form of size m over n variables where each clause contains at most one positive literal and another integer $k \leq n$.

Task: How many satisfying assignments are there for ϕ of weight $\leq k$?

Parameter: $k.log(n)$

Chapter 5

Conclusion

In this concluding chapter we like to discuss some of the open questions related to classifications of problems with respect to compression and give an overview of future direction of research in this domain. But before that, we would like to give an alternate interpretation of $\#VC$-hierarchy. For that, we are going to show a result that is basically the application of the theorem proved by N. Livne [40] in our context.

Before that, we would like to discuss the notion of witnessing relation. In the introductory chapter, we have already discussed about the concept of *verification algorithm* of NP problem. We can easily observe that for any VC_i problem for some $i \geq 2$, there is a verification algorithm as any such VC_i problem is also a NP problem.

It is easy to understand that any such verification algorithm V defines the parametric problem L as follows: $L = \{\langle x, 1^{|y|}\rangle | \exists y V(\langle x, 1^{|y|}\rangle, y) = 1\}$. We also know that such a verification algorithm is not unique. In fact, any problem L in VC_i for some $i \geq 2$ has infinitely many verification algorithms. Every such algorithm induces a relation R. We have already defined this concept of parametric relation in chapter 4. This relation also well-defines L as follows: $L = \{\langle x, 1^{|y|}\rangle | \exists y (x, y, 1^{|y|}) \in R\}$. We refer such a relation as witnessing relation here (as used in [40]), and say that L is the parametric problem defined by the relation R.

Proposition 31. *L ($\subseteq \{\langle x, 1^n\rangle | x \in \{0,1\}^*, n \in \mathbb{N}\}$) be a parametric problem in VC_i for some $i \geq 2$. Suppose there exists a polynomial time computable function f : $\{\langle x, 1^{n_1}\rangle | x \in \{0,1\}^*, n_1 \in \mathbb{N}\} \rightarrow \{\langle y, 1^{n_2}\rangle | y \in \{0,1\}^*, n_2 \in \mathbb{N}\}$ where n_1 and n_2 are polynomially bounded to each other, such that:*

1. *$image(f) \subseteq L$.*
2. *f is 1-1.*

3. *f is honest (f is said to be honest if there exists some polynomial q such that for all x, $|x| \leq q(|f(x)|)$).*

4. *There exists another parametric problem S in VC_i such that $L \setminus image(f) \subseteq S$ and $image(f) \subseteq \bar{S}$.*

Then, L has a witnessing relation that is $\#VC_i$ -complete.

Proof. To prove the proposition, we are going to construct a new verification algorithm for the parametric problem L. The relation defined by this new verification algorithm essentially *embeds* the natural witnessing relation for DEPTH$_i$CIRCUITSAT (using similar technique as suggested in [40]), but still defining L. Since the problem #DEPTH$_i$CIRCUITSAT is $\#VC_i$-complete, it will show that L is also $\#VC_i$-complete.

Let us assume that V_L and V_S are the verification algorithms for L and S respectively. Let us also consider that C is some DEPTH$_i$CIRCUITSAT instance with $n^{'}$ number of variables. We define the following verification algorithm $V^{'}$ for L which accepts w ($|w| \leq n$) as a witness for $\langle x, 1^n \rangle$ if and only if one of the following conditions hold:

1. $w = \langle 0, w^{'} \rangle$, where $w^{'}$ is some trivial string polynomially bounded with $n^{'}$ and $f(\langle C, 1^{n^{'}} \rangle) = \langle x, 1^n \rangle$.

2. $w = \langle 1, \tau \rangle$ where $f(\langle C, 1^{n^{'}} \rangle) = \langle x, 1^n \rangle$ and τ is a satisfying assignment for C.

3. $w = (y, z)$ where $V_S(\langle x, 1^n \rangle, y) = 1$ and $V_L(\langle x, 1^n \rangle, z) = 1$.

Firstly, we will show that $V^{'}$ defines L. To prove this, we can observe that every instance in L is either in $image(f)$ or in S, and thus will be accepted by conditions 1 or 3 of $V^{'}$, respectively. On the other hand, every instance not in L is not in $image(f)$ and hence can not be accepted by conditions 1 and 2 of $V^{'}$. Condition 3 accepts only instances in L, so it does not play any role in this part.

To complete the proof, we have to prove that we can find the total number of witnesses of the DEPTH$_i$CIRCUITSAT instance C from the reduced L-instance, using the function f. We are now going to show that total number of witnesses of C, $\#R(\langle C, 1^{n^{'}} \rangle) = \#R_{V'}(f(\langle C, 1^{n^{'}} \rangle))$-1, where R is the natural witnessing relation for the parametric problem #DEPTH$_i$CIRCUITSAT and $R_{V'}$ is the relation induced by $V^{'}$. To see that the relation is true, we can observe that for every unsatisfiable C, the L-instance $f(\langle C, 1^{n^{'}} \rangle)$ is accepted by condition 1, and only by it. Since f is 1-1 such $f(\langle C, 1^{n^{'}} \rangle)$ will have exactly one witness under $R_{V'}$. For every satisfiable C, every satisfying assignment contributes exactly one witness to the L-instance $f(\langle C, 1^{n^{'}} \rangle)$ (by condition 2 of $V^{'}$),

and since f is 1-1, no other circuit is mapped to $f(\langle C, 1^{n'} \rangle)$, thus there are no other witnesses contributed by condition 2 of V'. The first condition contributes exactly one more witness (again, since f is 1-1). Condition 3 does not play any role here as it contributes no witness (since $f(\langle C, 1^{n'} \rangle)$ is not in S). □

From the above result, we find an alternate way to prove the completeness results in $\#VC$ counting hierarchy. We have not done much work in this direction. But it can be considered a promising future direction for further research to investigate more useful properties of these counting hierarchies.

We now like to discuss some of the interesting problems that we could not solve. Firstly, we like to mention that, the nature of complexity class VC_E (section 3.3) is not completely clear, e.g. relation between VC_{OR} and VC_E is not known. One can try to investigate further in this direction to answer some interesting questions in this context. For example, one can try to prove if $VC_0 = VC_E$, some of the strong complexity theoretic assumptions are violated.

In chapter 4 we have proved that #DEPTH$_i$ MONOTONE CIRCUITSAT is $\#VC_i$-complete for any $i \geq 3$. same result for #DEPTH$_i$ ANTIMONOTONE CIRCUITSAT is also true. But we do not know whether it is correct for $i = 2$. To be more specific, let us consider the following two problems.

#MONOTONE SAT:

Input: A Boolean formula ϕ in CNF over n variables where all the literals are positive literals.

Task: How many satisfying assignments are there for ϕ ?

Parameter: n

#ANTIMONOTONE SAT:

Input: A Boolean formula ϕ in CNF over n variables where all the literals are negative literals.

Task: How many satisfying assignments are there for ϕ ?

Parameter: n

It is easy to check that both the above mentioned problems are equivalent with respect to w-T-reduction (Theorem 37, chapter 4). Corresponding decision problems are easy to solve and present in VC_0. But the counting versions are only known to be present in $\#VC_2$ (as any #MONOTONE SAT and #ANTIMONOTONE SAT instances are SAT instances). Finding some hardness result will be quite interesting in this context. We can even consider the following problem.

#HORNSAT:

Input: A Boolean formula ϕ in *CNF* over n variables where any clause contains at most one positive literal.

Task: How many satisfying assignments are there for ϕ ?

Parameter: n

HORNSAT is known to be P-complete and corresponding counting version is #P-complete. But once again, #HORNSAT is only known to be present in $\#VC_2$ without any known hardness result with respect to $\#VC$-hierarchy. It is easy to observe that any hardness result for #MONOTONE SAT or #ANTIMONOTONE SAT will prove hardness result for #HORNSAT as well.

We have already proved that $\#\text{KNAPSACK}_n$ is $\#VC_E$-hard in chapter 4. There is a scope of further improvement of this result. We have proved results related to #EXACT CNF-SAT and #EXACT DNF-SAT in chapter 4 (section 4.4 and 4.5). A few more results work for decision versions as mentioned there. If we extend those problems to depth k, similar techniques can be used to place them into VC_k. If one can prove those problems to be hard for VC_k (and $\#VC_k$ for counting problems), that will be really interesting. Not only this, there are many other related results one can try to improve specially in the counting hierarchy classifications (e.g. results related to $\#\text{EXACT TOUR}_n$ problem in chapter 4, section 4.4).

Finally, we think that we have just scratched the surface of a very interesting research topic. In future, we hope to see many other interesting results connecting this notion of instance compression with parametrized complexity, structural complexity, cryptography and other fields that we did not consider.

Bibliography

[1] A. Ben-Dor and S. Halevi. Zero-one permanent is #P-complete, a simpler proof. In Proceedings of the 2nd Israel Symposium on Theory of Computing Systems - ISTCS '93. Pages 108-117. IEEE, 1993.

[2] A. Drucker. New Limits to Classical and Quantum Instance Compression. IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS), pages 609-618, 2012.

[3] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. Journal of the ACM, Volume 28, Issue 1, pp. 114-133, 1981.

[4] A. Kolmogorov. On Tables of Random Numbers. Theoretical Computer Science 207 (2): 387395, 1998. doi:10.1016/S0304-3975(98)00075-9. MR 1643414.

[5] A. Maruoka. Concise Guide to Computation Theory. Springer, 2011.

[6] A. Shamir. IP = PSPACE. Journal of the ACM, 39(4):869-877, 1992.

[7] B. Dubrov and Y. Ishai. On the randomness complexity of efficient sampling. In 38th ACM Symposium on the Theory of Computing, pages 711-720, 2006.

[8] C. Berge, Two theorems in graph theory, Proc. Nat. Acad. Sci. U.S.A., 43 (1957), pp. 842-844.

[9] C. Chakraborty, R. Santhanam. Instance Compression for the Polynomial Hierarchy and Beyond, Proceeding of 7th International Symposium on Parameterized and Exact Computation (IPEC 2012), LNCS, pages 120-134, 2012.

[10] C. K. Yap. Some consequences of non-uniform conditions on uniform classes. Theoretical Computer Science, 26: 287-300, 1983.

[11] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. Journal of the ACM, 39(4):859-868, 1992.

[12] D.C. Kozen. The design and analysis of algorithms, chapter 26: Counting problems and #P, pages 138-143. Springer-Verlag, 1992.

[13] D. Harnik and M. Naor. On the compressibility of NP instances and cryptographic applications. In Proceedings if the 47th Annual IEEE Symposium on Foundations of Computer Science, pages 719-728, 2006.

[14] D. Hermelin, S. Kratsch, K. Soltys, M. Wahlstrom and X. Wu. Hierarchies of Inefficient Kernelizability. arXiv:1110.0976v1 [cs.CC].

[15] E. Tardos and J. Kleinberg. Algorithm Design. Addison-Wesley 2005. ISBN 0-321-29535-8.

[16] G. A. Jones, J. M. Jones. Elementary Number Theory. Springer 1998.

[17] H. Bodlaender, Rod Downey, Michael Fellows, and Danny Hermelin. On problems without polynomial kernels. In Proceedings of 35th International Colloquium on Automata, Languages and Programming, pages 563-574, 2008.

[18] H. Buhrman, J. M. Hitchcock: NP-Hard Sets are Exponentially Dense Unless NP is contained in CoNP/poly. Electronic Colloquium on Computational Complexity (ECCC) 15(022): (2008).

[19] H. L. Bodlaender. A cubic kernel for feedback vertex set. In Proceedings of the 24th annual conference on Theoretical aspects of computer science (STACS'07), Wolfgang Thomas and Pascal Weil (Eds.). Springer-Verlag, Berlin, Heidelberg, 320-331.

[20] H. W. Kuhn, The Hungari The Hungarian methodjbr the assignment problem, Naval Res. Logist. Quart., 2 (1955), pp. 83-97.

[21] J. Buss and J. Goldsmith. Nondeterminism within P. SIAM Journal on Computing. 22:560-572, 1993.

[22] J. E. Hopcroft, R. M. Karp: An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. 2(4): 225-231 (1973).

[23] J. E. Savage. Models of Computation: Exploring the Power of Computing. Addison Wesley Publishing Company, 1997.

[24] J. Flum and M. Grohe. Describing parameterized complexity classes. Information and Computation 187, 291-319 2003.

[25] J. Flum and M. Grohe. Parameterized Complexity Theory. Springer-Verlag New York, Inc., 2006.

[26] J. Flum and M. Grohe. The Parameterized Complexity of Counting Problems. SIAM J. Comput. 33(4): 892-922 (2004).

[27] J. Hartmanis, New Developments in Structural Complexity Theory (invited lecture), Proc. 15th International Colloquium on Automata, Languages and Programming, 1988 (ICALP 88), Lecture Notes in Computer Science, vol. 317 (1988), pp. 271-286.

[28] K.A. Abrahamson, R.G. Downey, and M.R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs. Annals of pure and applied logic, 73:235-276, 1995.

[29] L. Adleman. Two theorems on random polynomial time. In Proceedings of the 20th Annual IEEE Symposium on the Foundations of Computer Science, pages 75-83, 1978.

[30] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. Journal of Computer and System Sciences, 36: p.254-276. 1988.

[31] L. Cai, J. Chen, R. Downey, and M. Fellows. Advice classes of parameterized tractability. Annals of Pure and Applied logic, 84(1):119-138, 1997.

[32] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. Journal of Computer and System Sciences, 77(1):91-106, January 2011. Special issues celebrating Karp's Kyoto Prize.

[33] L. G. Valiant. The Complexity of Computing the Permanent. Theoretical Computer Science, Vol. 8 (1979), North-Holland Publishing Company, pp. 189-201.

[34] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. SIAM J. Comput. 8(3): 410-421 (1979).

[35] L. R. Ford and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N.J., 1965.

[36] M. Cesati. Perfect Code is W[1]-complete. Inf. Process. Lett. 81(3): 163-168 (2002).

[37] M. Hall, Distinct representatives of subsets, Bull. Amer. Math. Soc., 54 (1948), pp. 922-926.

[38] M. R. Fellows, D. Hermelin, F. A. Rosamond, S. Vialette: On the parameterized complexity of multiple-interval graph problems. Theor. Comput. Sci. 410(1): 53-61 (2009).

[39] M. Sipser. Introduction to the Theory of Computation (ISBN 0-534-95097-3). Course Technology, 2nd edition, 2005.

[40] N. Livne. A note on #P-completeness of NP-witnessing relations. Inf. Process. Lett. 109(5): 259-261 (2009).

[41] R. G. Downey, D. M. Thilikos. Confronting intractability via parameters. Computer Science Review 5(4): 279-317 (2011)

[42] R.G. Downey, M. R. Fellows. Fixed-parameter tractability and completeness I: basic results. SIAM J. Comput., 24 (1995), pp. 873921.

[43] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. Theor. Comp. Sc., 141:109-131, 1995.

[44] R. Downey and M. R. Fellows. Parameterized Complexity. Springer-Verlag, 1999.

[45] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum. pp. 85-103, 1972.

[46] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, pp. 302-309, doi:10.1145/800141.804678, 1980.

[47] R. Niedermeier. Invitation to Fixed Parameter Algorithms. Oxford University Press, 2006.

[48] S. Arora and B. Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

[49] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. Journal of the ACM, 45(3):501-555, 1998.

[50] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. Journal of the ACM, 45(1):70122, 1998.

[51] S. Bessy , F. V. Fomin , S. Gaspers , C. Paul , A. Perez , S. Saurabh , S. Thomasse. Kernels for Feedback Arc Set In Tournaments. Journal of Computer and System Sciences, Volume 77, Issue 6, November 2011, Pages 1071-1078.

[52] S. Cook. The complexity of theorem proving procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151158, 1971.

[53] S. Dziembowski. On forward-secure storage. In Advances in Cryptology - CRYPTO '06, Lecture Notes in Computer Science, volume 4117, pages 251-270. Springer, 2006.

[54] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge complexity of interactive proof-systems. Proceedings of 17th ACM Symposium on the Theory of Computation, Providence, Rhode Island. 1985, pp. 291-304.

[55] S. Kratsch, M. Wahlstrom: Preprocessing of Min Ones Problems: A Dichotomy. ICALP (1) 2010: 653-665.

[56] U. Manber. Introduction to algorithms - a creative approach. Addison-Wesley 1989, ISBN 978-0-201-12037-0, pp. I-XIV, 1-478.

[57] Y. Chen, J. Flum, M. Muller. Lower bounds for kernelizations. CRM Publications, Nov. 2008.

[58] Y. Chen and M. Muller. SAT is unlikely to be compressible. Manuscript, 2007.